

INTERNATIONAL
STANDARD

ISO/IEC/
IEEE
9945

First edition
2009-09-15

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

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

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



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

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

PDF disclaimer

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

Adobe is a trademark of Adobe Systems Incorporated.

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

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



COPYRIGHT PROTECTED DOCUMENT

© IEEE 2001-2008

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

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York • NY 10016-5997, USA
E-mail stds.ipr@ieee.org
Web www.ieee.org

Published in Switzerland

Foreword

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

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

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

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

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

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

(blank page)

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



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

Base Specifications, Issue 7

IEEE Computer Society

Sponsored by the
Portable Applications Standards Committee
and
The Open Group

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

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

(blank page)

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

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

The Open Group Technical Standard
Base Specifications, Issue 7

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

Sponsor

Portable Applications Standards Committee

of the

IEEE Computer Society

and

The Open Group

Approved 26 September 2008

IEEE-SA Standards Board

Approved 24 July 2008

The Open Group



Abstract

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

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

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

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

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

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

Keywords

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

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

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

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

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

PDF: ISBN 978-0-7381-5798-6 STD95820

CDROM: ISBN 978-0-7381-5799-3 STDCD95820

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

Doc. Number: C082

ISBN: 1-931624-79-8

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

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

IEEE

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

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

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

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

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

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

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

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

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

^A For this standard please send comments via the Austin Group, as indicated on page ii.

The Open Group

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

Further information on The Open Group is available at www.opengroup.org.

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

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

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

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

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at www.opengroup.org/corrigenda.

Introduction

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

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

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

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

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

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

Background

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

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

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

Audience

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

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

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

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

Purpose

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

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

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

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

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

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

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

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

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

POSIX.1-2008

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

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

Base Definitions

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

This volume is structured as follows:

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

Comprehensive references are available in the index.

System Interfaces

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

This volume is structured as follows:

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

Comprehensive references are available in the index.

Shell and Utilities

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

This volume is structured as follows:

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

Comprehensive references are available in the index.

Rationale (Informative)

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

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

Within this volume, the following terms are used:

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

Typographical Conventions

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

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

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

Note that:

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

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

Shading is used to identify extensions and options.

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

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

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

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

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

Notice to Users

Laws and Regulations

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

Copyrights

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

Updating of IEEE Documents

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

For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA web site at standards.ieee.org.

Errata

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

Feedback

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

Interpretations

Current interpretations can be accessed at the following web site: standards.ieee.org/reading/ieee/interp.

Patents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Contents

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

Trademarks

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

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

754TM, 854TM, 1003.0TM, 1003.1TM, 1003.1dTM, 1003.1gTM, 1003.1jTM, 1003.1qTM, 1003.2TM, 1003.2aTM, 1003.2dTM, 1003.9TM, and 1003.13TM are trademarks of the Institute of Electrical and Electronic Engineers, Inc.

AIX[®] is a registered trademark of IBM Corporation.

AT&T[®] is a registered trademark of AT&T in the USA and other countries.

Boundaryless Information FlowTM and TOGAFTM are trademarks and Motif[®], Making Standards Work[®], OSF/1[®], The Open Group[®], UNIX[®], and the “X” device are registered trademarks of The Open Group in the United States and other countries.

BSDTM is a trademark of the University of California, Berkeley, USA.

Hewlett-Packard[®], HP[®], and HP-UX[®] are registered trademarks of Hewlett-Packard Company.

IBM[®] is a registered trademark of International Business Machines Corporation.

IEEE[®] is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

Linux[®] is a registered trademark of Linus Torvalds.

POSIX[®] is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

Sun[®] and Sun Microsystems[®] are registered trademarks of Sun Microsystems, Inc.

/usr/group[®] is a registered trademark of UniForum, the International Network of UNIX System Users.

Acknowledgements

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

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

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

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

Referenced Documents

Normative References

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

Informative References

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

1984 /usr/group Standard

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

Almasi and Gottlieb

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

ANSI C

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

ANSI X3.226-1994

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

Brawer

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

DeRemer and Pennello Article

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

Draft ANSI X3J11.1

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

FIPS 151-1

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

FIPS 151-2

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

HP-UX Manual

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

IEC 60559: 1989

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

IEEE Standards Terms

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

Referenced Documents

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

Referenced Documents

IETF RFC 2181

Clarifications to the DNS Specification, R. Elz, R. Bush, July 1997 (available at: www.ietf.org/rfc/rfc2181.txt).

IETF RFC 2373

Internet Protocol, Version 6 (IPv6) Addressing Architecture, S. Deering, R. Hinden, July 1998 (available at: www.ietf.org/rfc/rfc2373.txt).

IETF RFC 2460

Internet Protocol, Version 6 (IPv6), S. Deering, R. Hinden, December 1998 (available at: www.ietf.org/rfc/rfc2460.txt).

Internationalisation Guide

Guide, July 1993, Internationalisation Guide, Version 2 (ISBN: 1-859120-02-4, G304), published by The Open Group.

ISO 2375:1985

ISO 2375:1985, Data Processing — Procedure for Registration of Escape Sequences.

ISO 8652:1987

ISO 8652:1987, Programming Languages — Ada (technically identical to ANSI standard 1815A-1983).

ISO/IEC 1539:1991

ISO/IEC 1539:1991, Information Technology — Programming Languages — Fortran (technically identical to the ANSI X3.9-1978 standard [FORTRAN 77]).

ISO/IEC 4873:1991

ISO/IEC 4873:1991, Information Technology — ISO 8-bit Code for Information Interchange — Structure and Rules for Implementation.

ISO/IEC 6429:1992

ISO/IEC 6429:1992, Information Technology — Control Functions for Coded Character Sets.

ISO/IEC 6937:1994

ISO/IEC 6937:1994, Information Technology — Coded Graphic Character Set for Text Communication — Latin Alphabet.

ISO/IEC 8802-3:1996

ISO/IEC 8802-3:1996, Information Technology — Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.

ISO/IEC 8859

ISO/IEC 8859, Information Technology — 8-Bit Single-Byte Coded Graphic Character Sets:

- Part 1: Latin Alphabet No. 1
- Part 2: Latin Alphabet No. 2
- Part 3: Latin Alphabet No. 3
- Part 4: Latin Alphabet No. 4
- Part 5: Latin/Cyrillic Alphabet
- Part 6: Latin/Arabic Alphabet
- Part 7: Latin/Greek Alphabet
- Part 8: Latin/Hebrew Alphabet
- Part 9: Latin Alphabet No. 5
- Part 10: Latin Alphabet No. 6
- Part 11: Latin/Thai Alphabet

Referenced Documents

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

ISO/IEC 9899: 1990

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

ISO POSIX-1: 1996

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

ISO POSIX-2: 1993

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

Issue 1

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

Issue 2

X/Open Portability Guide, January 1987:

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

Issue 3

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

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

Issue 4

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

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

Referenced Documents

Issue 4, Version 2

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

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

Issue 5

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

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

Issue 6

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

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

Knuth Article

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

KornShell

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

MSE Working Draft

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

POSIX.0: 1995

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

POSIX.1: 1988

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

POSIX.1: 1990

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

POSIX.1a

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

POSIX.1d: 1999

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

Referenced Documents

Amendment 4: Additional Realtime Extensions [C Language].

POSIX.1g: 2000

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

POSIX.1j: 2000

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

POSIX.1q: 2000

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

POSIX.2b

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

POSIX.2d: 1994

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

POSIX.13: 1998

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

Sarwate Article

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

Sprunt, Sha, and Lehoczky

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

SVID, Issue 1

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

SVID, Issue 2

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

SVID, Issue 3

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

The AWK Programming Language

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

UNIX Programmer's Manual

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

Referenced Documents

XNS, Issue 4

CAE Specification, August 1994, Networking Services, Issue 4 (ISBN: 1-85912-049-0, C438), published by The Open Group.

XNS, Issue 5

CAE Specification, February 1997, Networking Services, Issue 5 (ISBN: 1-85912-165-9, C523), published by The Open Group.

XNS, Issue 5.2

Technical Standard, January 2000, Networking Services (XNS), Issue 5.2 (ISBN: 1-85912-241-8, C808), published by The Open Group.

X/Open Curses, Issue 4, Version 2

CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), published by The Open Group.

Yacc

Yacc: Yet Another Compiler Compiler, Stephen C. Johnson, 1978.

Source Documents

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

AIX 3.2 Manual

AIX Version 3.2 For RISC System/6000, Technical Reference: Base Operating System and Extensions, 1990, 1992 (Part No. SC23-2382-00).

OSF/1

OSF/1 Programmer's Reference, Release 1.2 (ISBN: 0-13-020579-6).

OSF AES

Application Environment Specification (AES) Operating System Programming Interfaces Volume, Revision A (ISBN: 0-13-043522-8).

System V Release 2.0

- UNIX System V Release 2.0 Programmer's Reference Manual (April 1984 - Issue 2).
- UNIX System V Release 2.0 Programming Guide (April 1984 - Issue 2).

System V Release 4.2

Operating System API Reference, UNIX[®] SVR4.2 (1992) (ISBN: 0-13-017658-3).

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

Technical Standard: Base Specifications, Issue 7

Prepared by the Austin Group (www.opengroup.org/austin).

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

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

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

Technical Standard

Vol. 1:

Base Definitions, Issue 7

The Open Group

The Institute of Electrical and Electronics Engineers, Inc.

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

(blank page)

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

Introduction

1.1 Scope

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

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

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

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

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

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

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

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

- The Open Group Technical Standard, 2006, Extended API Set Part 1
- The Open Group Technical Standard, 2006, Extended API Set Part 2
- The Open Group Technical Standard, 2006, Extended API Set Part 3
- The Open Group Technical Standard, 2006, Extended API Set Part 4
- ISO/IEC 9899:1999, Programming Languages — C, including ISO/IEC 9899:1999/Cor.1:2001(E), ISO/IEC 9899:1999/Cor.2:2004(E), and ISO/IEC 9899:1999/Cor.3

Emphasis has been placed on standardizing existing practice for existing users, with changes and additions limited to correcting deficiencies in the following areas:

- Issues raised by Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1, and ISO/IEC defect reports against ISO/IEC 9945
- Issues raised in corrigenda for The Open Group Technical Standards and working group resolutions from The Open Group
- Issues arising from ISO TR 24715:2006, Conflicts between POSIX and the LSB
- Changes to make the text self-consistent with the additional material merged
- Features, marked Legacy or obsolescent in the base documents, have been considered for removal in this version
- A review and reorganization of the options within the standard
- Alignment with the ISO/IEC 9899:1999 standard, including ISO/IEC 9899:1999/Cor.2:2004(E)

1.2 Conformance

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

1.3 Normative References

The following standards contain provisions which, through references in POSIX.1-2008, constitute provisions of POSIX.1-2008. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on POSIX.1-2008 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ANS X3.9-1978

(Reaffirmed 1989) American National Standard for Information Systems: Standard X3.9-1978, Programming Language FORTRAN.¹

ISO/IEC 646:1991

ISO/IEC 646:1991, Information Processing — ISO 7-Bit Coded Character Set for Information Interchange.²

1. ANSI documents can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY 10018, USA.

*Introduction**Normative References*

ISO 4217: 2001

ISO 4217: 2001, Codes for the Representation of Currencies and Funds.

ISO 8601: 2004

ISO 8601: 2004, Data Elements and Interchange Formats — Information Interchange — Representation of Dates and Times.

ISO C (1999)

ISO/IEC 9899: 1999, Programming Languages — C, including ISO/IEC 9899: 1999/Cor.1: 2001(E), ISO/IEC 9899: 1999/Cor.2: 2004(E), and ISO/IEC 9899: 1999/Cor.3.

ISO/IEC 10646-1: 2000

ISO/IEC 10646-1: 2000, Information Technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.

1.4 Change History

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

1.5 Terminology

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

can

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

implementation-defined

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

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

legacy

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

may

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

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

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

shall

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

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

should

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

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

undefined

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

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

unspecified

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

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

1.6 Definitions and Concepts

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

1.7 Portability

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

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

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

system is any guide to its behavior on another system.

1.7.1 Codes

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

ADV Advisory Information

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

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

BE Batch Environment Services and Utilities

The functionality described is optional.

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

CD C-Language Development Utilities

The functionality described is optional.

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

CPT Process CPU-Time Clocks

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

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

CX Extension to the ISO C standard

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

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

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

FD FORTRAN Development Utilities

The functionality described is optional.

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

FR FORTRAN Runtime Utilities

The functionality described is optional.

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

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

Introduction

Portability

Where additional semantics apply to a function, the material is identified by use of the MSG margin legend.

MX IEC 60559 Floating-Point

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

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

OB Obsolescent

The functionality described may be removed in a future version of this volume of POSIX.1-2008. Strictly Conforming POSIX Applications and Strictly Conforming XSI Applications shall not use obsolescent features.

Where applicable, the material is identified by use of the OB margin legend.

OF Output Format Incompletely Specified

The functionality described is an XSI extension. The format of the output produced by the utility is not fully specified. It is therefore not possible to post-process this output in a consistent fashion. Typical problems include unknown length of strings and unspecified field delimiters.

Where applicable, the material is identified by use of the OF margin legend.

OH Optional Header

In the SYNOPSIS section of some interfaces in the System Interfaces volume of POSIX.1-2008 an included header is marked as in the following example:

```
#include <sys/types.h>
#include <grp.h>
struct group *getgrnam(const char *name);
```

The OH margin legend indicates that the marked header is not required on XSI-conformant systems.

PIO Prioritized Input and Output

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

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

PS Process Scheduling

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

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

RPI Robust Mutex Priority Inheritance

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

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

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

*Introduction**Portability*

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

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

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

XSI STREAMS

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

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

1.7.2 Margin Code Notation

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

A Feature Dependent on One or Two Options

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

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

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

A Feature Dependent on Either of the Options Denoted

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

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

A Feature Dependent on More than Two Options

The following shorthand notations are used:

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

Large Sections Dependent on an Option

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

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

(blank page)

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

2.1 Implementation Conformance

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

2.1.1 Requirements

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

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

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

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

2.1.2 Documentation

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

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

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

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

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

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

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

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

2.1.3 POSIX Conformance

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

Conformance

Implementation Conformance

2.1.3.1 POSIX System Interfaces

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

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

The following symbolic constants shall be defined with a value other than `-1`. If a constant is defined with the value zero, applications should use the `sysconf()`, `pathconf()`, or `fpathconf()` functions, or the `getconf` utility, to determine which features are present on the system at that time or for the particular pathname in question.

— `_POSIX_CHOWN_RESTRICTED`

The use of `chown()` is restricted to a process with appropriate privileges, and to changing the group ID of a file only to the effective group ID of the process or to one of its supplementary group IDs.

— `_POSIX_NO_TRUNC`

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

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

— Symbolic constants defined with the value 200809L:

`_POSIX_ASYNCHRONOUS_IO`

`_POSIX_BARRIERS`

`_POSIX_CLOCK_SELECTION`

`_POSIX_MAPPED_FILES`

`_POSIX_MEMORY_PROTECTION`

`_POSIX_READER_WRITER_LOCKS`

`_POSIX_REALTIME_SIGNALS`

`_POSIX_SEMAPHORES`

`_POSIX_SPIN_LOCKS`

`_POSIX_THREAD_SAFE_FUNCTIONS`

`_POSIX_THREADS`

`_POSIX_TIMEOUTS`

`_POSIX_TIMERS`

— Symbolic constants defined with a value greater than zero:

`_POSIX_JOB_CONTROL`

`_POSIX_SAVED_IDS`

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

`_POSIX_VDISABLE`

Note: The symbols above represent historical options that are no longer allowed as options, but are retained here for backwards-compatibility of applications.

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

```

_POSIX_ADVISORY_INFO
_POSIX_CPUTIME
_POSIX_FSYNC
_POSIX_IPV6
_POSIX_MEMLOCK
_POSIX_MEMLOCK_RANGE
_POSIX_MESSAGE_PASSING
_POSIX_MONOTONIC_CLOCK
_POSIX_PRIORITIZED_IO
_POSIX_PRIORITY_SCHEDULING
_POSIX_RAW_SOCKETS
_POSIX_SHARED_MEMORY_OBJECTS
_POSIX_SPAWN
_POSIX_SPORADIC_SERVER
_POSIX_SYNCHRONIZED_IO
_POSIX_THREAD_ATTR_STACKADDR
_POSIX_THREAD_CPUTIME
_POSIX_THREAD_ATTR_STACKSIZE
_POSIX_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING
_POSIX_THREAD_PROCESS_SHARED
_POSIX_THREAD_SPORADIC_SERVER
_POSIX_TRACE
_POSIX_TRACE_EVENT_FILTER
_POSIX_TRACE_INHERIT
_POSIX_TRACE_LOG
_POSIX_TYPED_MEMORY_OBJECTS
_XOPEN_CRYPT
_XOPEN_REALTIME
_XOPEN_REALTIME_THREADS
_XOPEN_STREAMS
_XOPEN_UNIX

```

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

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

2.1.3.2 POSIX Shell and Utilities

The following requirements apply to the shell and utilities:

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

*Conformance**Implementation Conformance*

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

```

POSIX2_C_DEV
POSIX2_CHAR_TERM
POSIX2_FORT_DEV
POSIX2_FORT_RUN
POSIX2_LOCALEDEF
POSIX2_PBS
POSIX2_PBS_ACCOUNTING
POSIX2_PBS_LOCATE
POSIX2_PBS_MESSAGE
POSIX2_PBS_TRACK
POSIX2_SW_DEV
POSIX2_UPE
XOPEN_UNIX
XOPEN_UUCP

```

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

2.1.4 XSI Conformance

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

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

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

2.1.4.1 XSI System Interfaces

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

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

```

_POSIX_FSYNC
_POSIX_THREAD_ATTR_STACKADDR
_POSIX_THREAD_ATTR_STACKSIZE
_POSIX_THREAD_PROCESS_SHARED

```

- The system may support the following XSI Option Groups (see [Section 2.1.5.2](#), on page 22) defined within POSIX.1-2008:

- Encryption
- Realtime
- Advanced Realtime
- Realtime Threads
- Advanced Realtime Threads
- Tracing
- XSI STREAMS

2.1.4.2 XSI Shell and Utilities Conformance

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

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

<code>admin</code>	<code>delta</code>	<code>rmdel</code>	<code>val</code>
<code>cflow</code>	<code>get</code>	<code>sact</code>	<code>what</code>
<code>ctags</code>	<code>nm</code>	<code>secs</code>	
<code>cxref</code>	<code>prs</code>	<code>unget</code>	

2.1.5 Option Groups

An Option Group is a group of related functions or options defined within the System Interfaces volume of POSIX.1-2008.

If an implementation supports an Option Group, then the system shall support the functional behavior described herein.

If an implementation does not support an Option Group, then the system need not support the functional behavior described herein.

2.1.5.1 Subprofiling Considerations

Profiling standards supporting functional requirements less than that required in POSIX.1-2008 may subset both mandatory and optional functionality required for POSIX Conformance (see [Section 2.1.3](#), on page 16) or XSI Conformance (see [Section 2.1.4](#), on page 19). Such profiles shall organize the subsets into Subprofiling Option Groups.

Conformance

Implementation Conformance

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

A Subprofiling Option Group may provide basic system functionality that other Subprofiling Option Groups and other options depend upon.³ If a profile of POSIX.1-2008 does not require an implementation to provide a Subprofiling Option Group that provides features utilized by a required Subprofiling Option Group (or option),⁴ the profile shall specify⁵ all of the following:

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

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

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

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

For the limits in **<limits.h>** and **<stdint.h>**:

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

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

-
3. As an example, the File System profiling option group provides underlying support for pathname resolution and file creation which are needed by any interface in POSIX.1-2008 that parses a *path* argument. If a profile requires support for the Device Input and Output profiling option group but does not require support for the File System profiling option group, the profile must specify how pathname resolution is to behave in that profile, how the **O_CREAT** flag to *open()* is to be handled (and the use of the character 'a' in the *mode* argument of *fopen()* when a filename argument names a file that does not exist), and specify lots of other details.
 4. As an example, POSIX.1-2008 requires that implementations claiming to support the Range Memory Locking option also support the Process Memory Locking option. A profile could require that the Range Memory Locking option had to be supplied without requiring that the Process Memory Locking option be supplied as long as the profile specifies everything an application developer or system implementor would have to know to build an application or implementation conforming to the profile.
 5. Note that the profile could just specify that any use of the features not specified by the profile would produce undefined or unspecified results.

— A profile may reduce a maximum acceptable value, but shall not make a maximum acceptable value larger.

- A profile shall not change a limit specified as having a minimum or maximum value into a limit specified as having a fixed value.
- A profile shall not create new limits.
- Any implementation that conforms to POSIX.1-2008 (including all options and extended limits required by the profile) shall also conform to that profile, except for the possible omission from `<unistd.h>` of a profile support indicator macro required by the profile.

2.1.5.2 XSI Option Groups

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

The following Option Groups are defined.

Encryption

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

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

These functions are marked CRYPT.

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

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

Realtime

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

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

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

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

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

*Conformance**Implementation Conformance*

774 _POSIX_FSYNC
 775 _POSIX_MEMLOCK
 776 _POSIX_MEMLOCK_RANGE
 777 _POSIX_MESSAGE_PASSING
 778 _POSIX_PRIORITIZED_IO
 779 _POSIX_PRIORITY_SCHEDULING
 780 _POSIX_SHARED_MEMORY_OBJECTS
 781 _POSIX_SYNCHRONIZED_IO

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

784 _POSIX_MEMLOCK
 785 _POSIX_MEMLOCK_RANGE
 786 _POSIX_MESSAGE_PASSING
 787 _POSIX_PRIORITY_SCHEDULING
 788 _POSIX_SHARED_MEMORY_OBJECTS
 789 _POSIX_SYNCHRONIZED_IO

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

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

799 **Advanced Realtime**

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

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

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

808 _POSIX_ADVISORY_INFO
 809 _POSIX_CPUTIME
 810 _POSIX_MONOTONIC_CLOCK
 811 _POSIX_SPAWN
 812 _POSIX_SPORADIC_SERVER
 813 _POSIX_TYPED_MEMORY_OBJECTS

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

816 _POSIX_ADVISORY_INFO
 817 _POSIX_CPUTIME
 818 _POSIX_MONOTONIC_CLOCK
 819 _POSIX_SPAWN
 820 _POSIX_SPORADIC_SERVER
 821 _POSIX_TYPED_MEMORY_OBJECTS

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

825 **Realtime Threads**

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

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

830 _POSIX_THREAD_PRIO_INHERIT
 831 _POSIX_THREAD_PRIO_PROTECT
 832 _POSIX_THREAD_PRIORITY_SCHEDULING

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

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

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

839 _POSIX_THREAD_PRIO_INHERIT
 840 _POSIX_THREAD_PRIO_PROTECT
 841 _POSIX_THREAD_PRIORITY_SCHEDULING

842 **Advanced Realtime Threads**

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

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

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

851 _POSIX_THREAD_CPUTIME
 852 _POSIX_THREAD_SPORADIC_SERVER

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

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

*Conformance**Implementation Conformance*

`_POSIX_THREAD_CPUTIME`
`_POSIX_THREAD_SPORADIC_SERVER`

Tracing

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

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

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

`_POSIX_TRACE`
`_POSIX_TRACE_EVENT_FILTER`
`_POSIX_TRACE_LOG`
`_POSIX_TRACE_INHERIT`

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

`_POSIX_TRACE`
`_POSIX_TRACE_EVENT_FILTER`
`_POSIX_TRACE_LOG`
`_POSIX_TRACE_INHERIT`

XSI STREAMS

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

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

It includes the following functions:

<code>fattach()</code>	<code>ioctl()</code>
<code>fdetach()</code>	<code>isastream()</code>
<code>getmsg()</code>	<code>putmsg()</code>
<code>getpmsg()</code>	<code>putpmsg()</code>

and the `<stropts.h>` header.

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

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

2.1.6 Options

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

1. Option not supported for compilation.

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

2. Option always supported.

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

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

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

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

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

2.1.6.1 System Interfaces

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

Conformance

Implementation Conformance

2.1.6.2 Shell and Utilities

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

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

CD POSIX2_C_DEV

The system supports the C-Language Development Utilities option.

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

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

c99

lex

yacc

POSIX2_CHAR_TERM

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

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

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

FD POSIX2_FORT_DEV

The system supports the FORTRAN Development Utilities option.

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

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

FR POSIX2_FORT_RUN

The system supports the FORTRAN Runtime Utilities option.

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

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

POSIX2_LOCALEDEF

The system supports the Locale Creation Utilities option.

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

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

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

Conformance

Implementation Conformance

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

1030 2.2 Application Conformance

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

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

1037 2.2.1 Strictly Conforming POSIX Application

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

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

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

2.2.2 Conforming POSIX Application

2.2.2.1 ISO/IEC Conforming POSIX Application

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

2.2.2.2 <National Body> Conforming POSIX Application

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

2.2.3 Conforming POSIX Application Using Extensions

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

2.2.4 Strictly Conforming XSI Application

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

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

*Conformance**Application Conformance*

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

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

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

2.2.5 Conforming XSI Application Using Extensions

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

2.3 Language-Dependent Services for the C Programming Language

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

2.4 Other Language-Related Specifications

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

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

(blank page)

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

Definitions

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

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

3.1 Abortive Release

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

3.2 Absolute Pathname

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

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

3.3 Access Mode

A particular form of access permitted to a file.

3.4 Additional File Access Control Mechanism

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

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

3.5 Address Space

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

3.6 Advisory Information

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

3.7 Affirmative Response

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

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

3.8 Alert

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

3.9 Alert Character (<alert>)

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

3.10 Alias Name

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

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

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

*Definitions**Alignment***3.11 Alignment**

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

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

3.12 Alternate File Access Control Mechanism

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

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

3.13 Alternate Signal Stack

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

3.14 Ancillary Data

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

3.15 Angle Brackets

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

3.16 Apostrophe Character (<apostrophe>)

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

3.17 Application

A computer program that performs some desired function.

1196 3.18 Application Address

1197 Endpoint address of a specific application.

1198 3.19 Application Program Interface (API)

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

1200 3.20 Appropriate Privileges

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

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

1207 3.21 Argument

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

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

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

1215 3.22 Arm (a Timer)

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

1218 3.23 Asterisk Character (<asterisk>)

1219 The character ' * '.

1220 3.24 Async-Cancel-Safe Function

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

Definitions

Asynchronous Events

3.25 Asynchronous Events

Events that occur independently of the execution of the application.

3.26 Asynchronous Input and Output

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

3.27 Async-Signal-Safe Function

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

3.28 Asynchronously-Generated Signal

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

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

3.29 Asynchronous I/O Completion

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

3.30 Asynchronous I/O Operation

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

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

3.31 Authentication

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

3.32 Authorization

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

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

3.33 Background Job

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

3.34 Background Process

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

3.35 Background Process Group (or Background Job)

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

3.36 Backquote Character

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

3.37 Backslash Character (<backslash>)

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

3.38 Backspace Character (<backspace>)

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

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

*Definitions**Barrier***3.39 Barrier**

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

3.40 Basename

The final, or only, filename in a pathname.

3.41 Basic Regular Expression (BRE)

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

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

3.42 Batch Access List

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

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

3.43 Batch Administrator

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

3.44 Batch Client

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

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

3.45 Batch Destination

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

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

3.46 Batch Destination Identifier

A string that identifies a specific batch destination.

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

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

3.47 Batch Directive

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

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

3.48 Batch Job

A set of computational tasks for a computing system.

Batch jobs are managed by batch servers.

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

3.49 Batch Job Attribute

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

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

3.50 Batch Job Identifier

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

Definitions

Batch Job Name

3.51 Batch Job Name

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

3.52 Batch Job Owner

The *username@hostname* of the user submitting the batch job, where *username* is a user name (see also [Section 3.429](#), on page 102) and *hostname* is a network host name.

3.53 Batch Job Priority

A value specified by the user that may be used by an implementation to determine the order in which batch jobs are selected to be executed. Job priority has a numeric value in the range -1 024 to 1 023.

Note: The batch job priority is not the execution priority (nice value) of the batch job.

3.54 Batch Job State

An attribute of a batch job which determines the types of requests that the batch server that manages the batch job can accept for the batch job. Valid states include QUEUED, RUNNING, HELD, WAITING, EXITING, and TRANSITING.

3.55 Batch Name Service

A service that assigns batch names that are unique within the batch name space, and that can translate a unique batch name into the location of the named batch entity.

3.56 Batch Name Space

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

3.57 Batch Node

A host containing part or all of a batch system.

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

- Capable of executing a batch client
- Contains a routing batch queue
- Contains an execution batch queue

3.58 Batch Operator

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

3.59 Batch Queue

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

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

See also XCU [Section 3.1.2](#) (on page 2376).

3.60 Batch Queue Attribute

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

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

3.61 Batch Queue Position

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

3.62 Batch Queue Priority

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

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

Definitions

Batch Rerunability

3.63 Batch Rerunability

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

3.64 Batch Restart

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

3.65 Batch Server

A computational entity that provides batch services.

3.66 Batch Server Name

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

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

3.67 Batch Service

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

Batch services are of two types: requested and deferred.

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

3.68 Batch Service Request

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

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

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

3.69 Batch Submission

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

1390 **3.70 Batch System**

1391 A collection of one or more batch servers.

1392 **3.71 Batch Target User**

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

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

1396 **3.72 Batch User**

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

1398 **3.73 Bind**

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

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

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

1404 **3.75 Blank Line**

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

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

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

1410 **3.77 Blocking**

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

3.78 Block-Mode Terminal

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

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

3.79 Block Special File

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

3.80 Braces

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

3.81 Brackets

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

3.82 Broadcast

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

3.83 Built-In Utility (or Built-In)

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

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

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

3.84 Byte

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

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

3.85 Byte Input/Output Functions

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

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

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

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

3.87 Character

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

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

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

3.88 Character Array

An array of elements of type **char**.

3.89 Character Class

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

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

3.90 Character Set

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

3.91 Character Special File

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

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

3.92 Character String

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

3.93 Child Process

A new process created (by *fork()*, *posix_spawn()*, or *posix_spawnnp()*) by a given process. A child process remains the child of the creating process as long as both processes continue to exist.

Note: The *fork()*, *posix_spawn()*, and *posix_spawnnp()* functions are defined in detail in the System Interfaces volume of POSIX.1-2008.

3.94 Circumflex Character (<circumflex>)

The character ' ^ '.

3.95 Clock

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

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

3.96 Clock Jump

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

3.97 Clock Tick

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

3.98 Coded Character Set

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

3.99 Codeset

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

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

3.100 Collating Element

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

3.101 Collation

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

3.102 Collation Sequence

The relative order of collating elements as determined by the setting of the *LC_COLLATE* category in the current locale. The collation sequence is used for sorting and is determined from the collating weights assigned to each collating element. In the absence of weights, the collation sequence is the order in which collating elements are specified between **order_start** and **order_end** keywords in the *LC_COLLATE* category.

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

Note: {*COLL_WEIGHTS_MAX*} is defined in detail in [<limits.h>](#).

3.103 Column Position

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

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

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

3.104 Command

A directive to the shell to perform a particular task.

Note: Shell Commands are defined in detail in XCU [Section 2.9](#) (on page 2316).

3.105 Command Language Interpreter

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

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

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

3.106 Composite Graphic Symbol

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

3.107 Condition Variable

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

*Definitions**Connected Socket***3.108 Connected Socket**

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

3.109 Connection

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

3.110 Connection Mode

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

3.111 Connectionless Mode

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

3.112 Control Character

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

3.113 Control Operator

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

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

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

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

3.114 Controlling Process

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

3.115 Controlling Terminal

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

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

3.116 Conversion Descriptor

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

3.117 Core File

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

3.118 CPU Time (Execution Time)

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

3.119 CPU-Time Clock

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

3.120 CPU-Time Timer

A timer attached to a CPU-time clock.

3.121 Current Job

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

*Definitions**Current Working Directory***3.122 Current Working Directory**

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

3.123 Cursor Position

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

3.124 Datagram

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

3.125 Data Segment

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

3.126 Deferred Batch Service

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

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

3.127 Device

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

3.128 Device ID

A non-negative integer used to identify a device.

3.129 Directory

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

3.130 Directory Entry (or Link)

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

1630 with the same file.

1631 **3.131 Directory Stream**

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

1634 **3.132 Disarm (a Timer)**

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

1637 **3.133 Display**

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

1640 **3.134 Display Line**

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

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

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

1645 The character '\$'.

1646 **3.136 Dot**

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

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

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

Definitions

Dot-Dot

3.137 Dot-Dot

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

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

3.138 Double-Quote Character

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

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

3.139 Downshifting

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

3.140 Driver

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

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

3.141 Effective Group ID

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

3.142 Effective User ID

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

3.143 Eight-Bit Transparency

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

1676 3.144 Empty Directory

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

1681 3.145 Empty Line

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

1683 3.146 Empty String (or Null String)

1684 A string whose first byte is a null byte.

1685 3.147 Empty Wide-Character String

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

1687 3.148 Encoding Rule

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

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

1690 3.149 Entire Regular Expression

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

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

3.150 Epoch

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

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

3.151 Equivalence Class

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

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

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

3.152 Era

A locale-specific method for counting and displaying years.

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

3.153 Event Management

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

3.154 Executable File

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

1716 **3.155 Execute**

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

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

1720 **3.156 Execution Time**

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

1722 **3.157 Execution Time Monitoring**

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

1725 **3.158 Expand**

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

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

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

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

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

1732 **3.160 Extended Security Controls**

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

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

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

3.161 Feature Test Macro

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

Note: See also XSH [Section 2.2](#) (on page 468).

3.162 Field

In the shell command language, a unit of text that is the result of parameter expansion, arithmetic expansion, command substitution, or field splitting. During command processing, the resulting fields are used as the command name and its arguments.

Note: Parameter Expansion is defined in detail in XCU [Section 2.6.2](#) (on page 2306).

Arithmetic Expansion is defined in detail in XCU [Section 2.6.4](#) (on page 2310).

Command Substitution is defined in detail in XCU [Section 2.6.3](#) (on page 2309).

Field Splitting is defined in detail in XCU [Section 2.6.5](#) (on page 2311).

For further information on command processing, see XCU [Section 2.9.1](#) (on page 2316).

3.163 FIFO Special File (or FIFO)

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

Note: Other characteristics of FIFOs are described in the System Interfaces volume of POSIX.1-2008, *lseek()*, *open()*, *read()*, and *write()*.

3.164 File

An object that can be written to, or read from, or both. A file has certain attributes, including access permissions and type. File types include regular file, character special file, block special file, FIFO special file, symbolic link, socket, and directory. Other types of files may be supported by the implementation.

3.165 File Description

See *Open File Description* in [Section 3.253](#) (on page 73).

3.166 File Descriptor

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

Note: {OPEN_MAX} is defined in detail in [<limits.h>](#).

3.167 File Group Class

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

3.168 File Mode

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

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

3.169 File Mode Bits

A file's file permission bits, set-user-ID-on-execution bit (S_ISUID), set-group-ID-on-execution bit (S_ISGID), and, on directories, the restricted deletion flag bit (S_ISVTX).

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

3.170 Filename

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

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

3.171 File Offset

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

Definitions

File Offset

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

1792 **3.172 File Other Class**

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

1796 **3.173 File Owner Class**

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

1800 **3.174 File Permission Bits**

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

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

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

1807 **3.175 File Serial Number**

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

1809 **3.176 File System**

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

1812 **3.177 File Type**

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

1814 **3.178 Filter**

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

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

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

1820 **3.180 Flow Control**

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

1823 **3.181 Foreground Job**

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

1825 **3.182 Foreground Process**

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

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

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

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

1833 **3.184 Foreground Process Group ID**

1834 The process group ID of the foreground process group.

Definitions

Form-Feed Character (<form-feed>)

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

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

3.186 Graphic Character

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

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

3.187 Group Database

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

- Group name
- Numerical group ID
- List of users allowed in the group

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

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

3.188 Group ID

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

3.189 Group Name

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

3.190 Hard Limit

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

3.191 Hard Link

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

3.192 Home Directory

The directory specified by the *HOME* environment variable.

3.193 Host Byte Order

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

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

3.194 Incomplete Line

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

3.195 Inf

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

3.196 Instrumented Application

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

3.197 Interactive Shell

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

*Definitions**Internationalization***3.198 Internationalization**

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

3.199 Interprocess Communication

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

3.200 Invoke

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

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

3.201 Job

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

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

3.202 Job Control

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

3.203 Job Control Job ID

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

1908
1909
1910
1911
1912
1913
1914
1915
1916

1917

1918

1919

1920

1921

1922

1923

1924

1925

1926

1927

1928
1929

1930

1931

Table 3-1 Job Control Job ID Formats

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

3.204 Last Close (of a File)

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

3.205 Line

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

3.206 Linger

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

3.207 Link

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

3.208 Link Count

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

3.209 Local Customs

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

3.210 Local Interprocess Communication (Local IPC)

The transfer of data between processes in the same system.

Definitions

Locale

3.211 Locale

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

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

3.212 Localization

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

3.213 Login

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

3.214 Login Name

A user name that is associated with a login.

3.215 Map

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

3.216 Marked Message

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

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

3.217 Matched

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

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

3.218 Memory Mapped Files

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

3.219 Memory Object

One of:

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

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

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

3.220 Memory-Resident

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

3.221 Message

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

3.222 Message Catalog

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

Definitions

Message Catalog Descriptor

3.223 Message Catalog Descriptor

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

3.224 Message Queue

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

3.225 Mode

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

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

3.226 Monotonic Clock

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

3.227 Mount Point

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

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

3.228 Multi-Character Collating Element

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

3.229 Mutex

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

2006 **3.230 Name**

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

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

2010 **3.231 Named STREAM**

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

2014 **3.232 NaN (Not a Number)**

2015 A set of values that may be stored in a floating type but that are neither Inf nor valid floating-
 2016 point numbers. Not all systems support NaN values.

2017 **3.233 Native Language**

2018 A computer user's spoken or written language, such as American English, British English,
 2019 Danish, Dutch, French, German, Italian, Japanese, Norwegian, or Swedish.

2020 **3.234 Negative Response**

2021 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category
 2022 keyword **noexpr**, matching an extended regular expression in the current locale.

2023 **Note:** The *LC_MESSAGES* category is defined in detail in [Section 7.3.6](#) (on page 164).

2024 **3.235 Network**

2025 A collection of interconnected hosts.

2026 **Note:** The term "network" in POSIX.1-2008 is used to refer to the network of hosts. The term "batch
 2027 system" is used to refer to the network of batch servers.

2028 **3.236 Network Address**

2029 A network-visible identifier used to designate specific endpoints in a network. Specific
 2030 endpoints on host systems have addresses, and host systems may also have addresses.

3.237 Network Byte Order

The way of representing any integer type such that, when transmitted over a network via a network endpoint, the **int** type is transmitted as an appropriate number of octets with the most significant octet first, followed by any other octets in descending order of significance.

Note: This order is more commonly known as big-endian ordering. See also [Section 4.9](#) (on page 110).

3.238 Newline Character (<newline>)

A character that in the output stream indicates that printing should start at the beginning of the next line. It is the character designated by '`\n`' in the C language. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the next line.

3.239 Nice Value

A number used as advice to the system to alter process scheduling. Numerically smaller values give a process additional preference when scheduling a process to run. Numerically larger values reduce the preference and make a process less likely to run. Typically, a process with a smaller nice value runs to completion more quickly than an equivalent process with a higher nice value. The symbol {NZERO} specifies the default nice value of the system.

3.240 Non-Blocking

A property of an open file description that causes function calls involving it to return without delay when it is detected that the requested action associated with the function call cannot be completed without unknown delay.

Note: The exact semantics are dependent on the type of file associated with the open file description. For data reads from devices such as ttys and FIFOs, this property causes the read to return immediately when no data was available. Similarly, for writes, it causes the call to return immediately when the thread would otherwise be delayed in the write operation; for example, because no space was available. For networking, it causes functions not to await protocol events (for example, acknowledgements) to occur. See also XSH [Section 2.10.7](#) (on page 519).

3.241 Non-Spacing Characters

A character, such as a character representing a diacritical mark in the ISO/IEC 6937:2001 standard coded graphic character set, which is used in combination with other characters to form composite graphic symbols.

2061 **3.242 NUL**

2062 A character with all bits set to zero.

2063 **3.243 Null Byte**

2064 A byte with all bits set to zero.

2065 **3.244 Null Pointer**

2066 A pointer obtained by converting an integer constant expression with the value 0, or such an
 2067 expression cast to type **void ***, to a pointer type; for example, **(char *)0**. The C language
 2068 guarantees that a null pointer compares unequal to a pointer to any object or function, so it is
 2069 used by many functions that return pointers to indicate an error.

2070 **3.245 Null String**2071 See *Empty String* in [Section 3.146](#) (on page 56).2072 **3.246 Null Wide-Character Code**

2073 A wide-character code with all bits set to zero.

2074 **3.247 Number-Sign Character (<number-sign>)**

2075 The character ' # ' , also known as hash sign.

2076 **3.248 Object File**

2077 A regular file containing the output of a compiler, formatted as input to a linkage editor for
 2078 linking with other object files into an executable form. The methods of linking are unspecified
 2079 and may involve the dynamic linking of objects at runtime. The internal format of an object file
 2080 is unspecified, but a conforming application cannot assume an object file is a text file.

2081 **3.249 Octet**

2082 Unit of data representation that consists of eight contiguous bits.

*Definitions**Offset Maximum*2083 **3.250 Offset Maximum**

2084 An attribute of an open file description representing the largest value that can be used as a file
 2085 offset.

2086 **3.251 Opaque Address**

2087 An address such that the entity making use of it requires no details about its contents or format.

2088 **3.252 Open File**

2089 A file that is currently associated with a file descriptor.

2090 **3.253 Open File Description**

2091 A record of how a process or group of processes is accessing a file. Each file descriptor refers to
 2092 exactly one open file description, but an open file description can be referred to by more than
 2093 one file descriptor. The file offset, file status, and file access modes are attributes of an open file
 2094 description.

2095 **3.254 Operand**

2096 An argument to a command that is generally used as an object supplying information to a utility
 2097 necessary to complete its processing. Operands generally follow the options in a command line.

2098 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213).

2099 **3.255 Operator**

2100 In the shell command language, either a control operator or a redirection operator.

2101 **3.256 Option**

2102 An argument to a command that is generally used to specify changes in the utility's default
2103 behavior.

2104 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213).

2105 **3.257 Option-Argument**

2106 A parameter that follows certain options. In some cases an option-argument is included within
2107 the same argument string as the option—in most cases it is the next argument.

2108 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213).

2109 **3.258 Orientation**

2110 A stream has one of three orientations: unoriented, byte-oriented, or wide-oriented.

2111 **Note:** For further information, see XSH [Section 2.5.2](#) (on page 493).

2112 **3.259 Orphaned Process Group**

2113 A process group in which the parent of every member is either itself a member of the group or is
2114 not a member of the group's session.

2115 **3.260 Page**

2116 The granularity of process memory mapping or locking.

2117 Physical memory and memory objects can be mapped into the address space of a process on
2118 page boundaries and in integral multiples of pages. Process address space can be locked into
2119 memory (made memory-resident) on page boundaries and in integral multiples of pages.

2120 **3.261 Page Size**

2121 The size, in bytes, of the system unit of memory allocation, protection, and mapping. On
2122 systems that have segment rather than page-based memory architectures, the term "page"
2123 means a segment.

*Definitions**Parameter***3.262 Parameter**

In the shell command language, an entity that stores values. There are three types of parameters: variables (named parameters), positional parameters, and special parameters. Parameter expansion is accomplished by introducing a parameter with the '\$' character.

Note: See also XCU [Section 2.5](#) (on page 2301).

In the C language, an object declared as part of a function declaration or definition that acquires a value on entry to the function, or an identifier following the macro name in a function-like macro definition.

3.263 Parent Directory

When discussing a given directory, the directory that both contains a directory entry for the given directory and is represented by the pathname dot-dot in the given directory.

When discussing other types of files, a directory containing a directory entry for the file under discussion.

This concept does not apply to dot and dot-dot.

3.264 Parent Process

The process which created (or inherited) the process under discussion.

3.265 Parent Process ID

An attribute of a new process identifying the parent of the process. The parent process ID of a process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime has ended, the parent process ID is the process ID of an implementation-defined system process.

3.266 Pathname

A character string that is used to identify a file. In the context of POSIX.1-2008, a pathname may be limited to {PATH_MAX} bytes, including the terminating null byte. It has an optional beginning <slash>, followed by zero or more filenames separated by <slash> characters. A pathname may optionally contain one or more trailing <slash> characters. Multiple successive <slash> characters are considered to be the same as one <slash>, except for the case of exactly two leading <slash> characters.

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

3.267 Pathname Component

See *Filename* in [Section 3.170](#) (on page 60).

3.268 Path Prefix

The part of a pathname up to, but not including, the last component and any trailing <slash> characters, unless the pathname consists entirely of <slash> characters, in which case the path prefix is ' / ' for a pathname containing either a single <slash> or three or more <slash> characters, and ' / / ' for the pathname // . The path prefix of a pathname containing no <slash> characters is empty, but is treated as referring to the current working directory.

Note: The term is used both in the sense of identifying part of a pathname that forms the prefix and of joining a non-empty path prefix to a filename to form a pathname. In the latter case, the path prefix need not have a trailing <slash> (in which case the joining is done with a <slash> character).

3.269 Pattern

A sequence of characters used either with regular expression notation or for pathname expansion, as a means of selecting various character strings or pathnames, respectively.

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

See also XCU [Section 2.6.6](#) (on page 2311).

The syntaxes of the two types of patterns are similar, but not identical; POSIX.1-2008 always indicates the type of pattern being referred to in the immediate context of the use of the term.

3.270 Period Character (<period>)

The character ' . ' . The term “period” is contrasted with dot (see also [Section 3.136](#), on page 54), which is used to describe a specific directory entry.

3.271 Permissions

Attributes of an object that determine the privilege necessary to access or manipulate the object.

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

3.272 Persistence

A mode for semaphores, shared memory, and message queues requiring that the object and its state (including data, if any) are preserved after the object is no longer referenced by any process.

*Definitions**Persistence*

2181 Persistence of an object does not imply that the state of the object is maintained across a system
 2182 crash or a system reboot.

2183 **3.273 Pipe**

2184 An object identical to a FIFO which has no links in the file hierarchy.

2185 **Note:** The *pipe()* function is defined in detail in the System Interfaces volume of POSIX.1-2008.

2186 **3.274 Polling**

2187 A scheduling scheme whereby the local process periodically checks until the pre-specified
 2188 events (for example, read, write) have occurred.

2189 **3.275 Portable Character Set**

2190 The collection of characters that are required to be present in all locales supported by
 2191 conforming systems.

2192 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 125).

2193 This term is contrasted against the smaller portable filename character set; see also [Section 3.276](#).

2194 **3.276 Portable Filename Character Set**

2195 The set of characters from which portable filenames are constructed.

2196 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 2197 a b c d e f g h i j k l m n o p q r s t u v w x y z
 2198 0 1 2 3 4 5 6 7 8 9 . _ -

2199 The last three characters are the <period>, <underscore>, and <hyphen> characters, respectively.

2200 **3.277 Positional Parameter**

2201 In the shell command language, a parameter denoted by a single digit or one or more digits in
 2202 curly braces.

2203 **Note:** For further information, see XCU [Section 2.5.1](#) (on page 2301).

2204 **3.278 Preallocation**

2205 The reservation of resources in a system for a particular use.

2206 Preallocation does not imply that the resources are immediately allocated to that use, but merely
 2207 indicates that they are guaranteed to be available in bounded time when needed.

2208 **3.279 Preempted Process (or Thread)**

2209 A running thread whose execution is suspended due to another thread becoming runnable at a
 2210 higher priority.

2211 **3.280 Previous Job**

2212 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities if the
 2213 current job exits. There is at most one previous job; see also [Section 3.203](#) (on page 65).

2214 **3.281 Printable Character**

2215 One of the characters included in the **print** character classification of the *LC_CTYPE* category in
 2216 the current locale.

2217 **Note:** The *LC_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 139).

2218 **3.282 Printable File**

2219 A text file consisting only of the characters included in the **print** and **space** character
 2220 classifications of the *LC_CTYPE* category and the <backspace>, all in the current locale.

2221 **Note:** The *LC_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 139).

Definitions

Priority

3.283 Priority

A non-negative integer associated with processes or threads whose value is constrained to a range defined by the applicable scheduling policy. Numerically higher values represent higher priorities.

3.284 Priority Band

The queuing order applied to normal priority STREAMS messages. High priority STREAMS messages are not grouped by priority bands. The only differentiation made by the STREAMS mechanism is between zero and non-zero bands, but specific protocol modules may differentiate between priority bands.

3.285 Priority Inversion

A condition in which a thread that is not voluntarily suspended (waiting for an event or time delay) is not running while a lower priority thread is running. Such blocking of the higher priority thread is often caused by contention for a shared resource.

3.286 Priority Scheduling

A performance and determinism improvement facility to allow applications to determine the order in which threads that are ready to run are granted access to processor resources.

3.287 Priority-Based Scheduling

Scheduling in which the selection of a running thread is determined by the priorities of the runnable processes or threads.

3.288 Privilege

See *Appropriate Privileges* in [Section 3.20](#) (on page 36).

2243 **3.289 Process**

2244 An address space with one or more threads executing within that address space, and the
2245 required system resources for those threads.

2246 **Note:** Many of the system resources defined by POSIX.1-2008 are shared among all of the threads
2247 within a process. These include the process ID, the parent process ID, process group ID, session
2248 membership, real, effective, and saved set-user-ID, real, effective, and saved set-group-ID,
2249 supplementary group IDs, current working directory, root directory, file mode creation mask,
2250 and file descriptors.

2251 **3.290 Process Group**

2252 A collection of processes that permits the signaling of related processes. Each process in the
2253 system is a member of a process group that is identified by a process group ID. A newly created
2254 process joins the process group of its creator.

2255 **3.291 Process Group ID**

2256 The unique positive integer identifier representing a process group during its lifetime.

2257 **Note:** See also *Process Group ID Reuse* defined in [Section 4.13](#) (on page 112).

2258 **3.292 Process Group Leader**

2259 A process whose process ID is the same as its process group ID.

2260 **3.293 Process Group Lifetime**

2261 The period of time that begins when a process group is created and ends when the last
2262 remaining process in the group leaves the group, due either to the end of the lifetime of the last
2263 process or to the last remaining process calling the *setsid()* or *setpgid()* functions.

2264 **Note:** The *setsid()* and *setpgid()* functions are defined in detail in the System Interfaces volume of
2265 POSIX.1-2008.

3.294 Process ID

The unique positive integer identifier representing a process during its lifetime.

Note: See also *Process ID Reuse* defined in [Section 4.13](#) (on page 112).

3.295 Process Lifetime

The period of time that begins when a process is created and ends when its process ID is returned to the system. After a process is created by *fork()*, *posix_spawn()*, or *posix_spawnp()*, it is considered active. At least one thread of control and address space exist until it terminates. It then enters an inactive state where certain resources may be returned to the system, although some resources, such as the process ID, are still in use. When another process executes a *wait()*, *waitid()*, or *waitpid()* function for an inactive process, the remaining resources are returned to the system. The last resource to be returned to the system is the process ID. At this time, the lifetime of the process ends.

Note: The *fork()*, *posix_spawn()*, *posix_spawnp()*, *wait()*, *waitid()*, and *waitpid()* functions are defined in detail in the System Interfaces volume of POSIX.1-2008.

3.296 Process Memory Locking

A performance improvement facility to bind application programs into the high-performance random access memory of a computer system. This avoids potential latencies introduced by the operating system in storing parts of a program that were not recently referenced on secondary memory devices.

3.297 Process Termination

There are two kinds of process termination:

1. Normal termination occurs by a return from *main()*, when requested with the *exit()*, *_exit()*, or *_Exit()* functions; or when the last thread in the process terminates by returning from its start function, by calling the *pthread_exit()* function, or through cancellation.
2. Abnormal termination occurs when requested by the *abort()* function or when some signals are received.

Note: The *_exit()*, *_Exit()*, *abort()*, and *exit()* functions are defined in detail in the System Interfaces volume of POSIX.1-2008.

3.298 Process-To-Process Communication

The transfer of data between processes.

2297 **3.299 Process Virtual Time**

2298 The measurement of time in units elapsed by the system clock while a process is executing.

2299 **3.300 Program**

2300 A prepared sequence of instructions to the system to accomplish a defined task. The term
 2301 “program” in POSIX.1-2008 encompasses applications written in the Shell Command Language,
 2302 complex utility input languages (for example, *awk*, *lex*, *sed*, and so on), and high-level languages.

2303 **3.301 Protocol**

2304 A set of semantic and syntactic rules for exchanging information.

2305 **3.302 Pseudo-Terminal**

2306 A facility that provides an interface that is identical to the terminal subsystem, except where
 2307 noted otherwise in POSIX.1-2008. A pseudo-terminal is composed of two devices: the “master
 2308 device” and a “slave device”. The slave device provides processes with an interface that is
 2309 identical to the terminal interface, although there need not be hardware behind that interface.
 2310 Anything written on the master device is presented to the slave as an input and anything
 2311 written on the slave device is presented as an input on the master side.

2312 **3.303 Radix Character**

2313 The character that separates the integer part of a number from the fractional part.

2314 **3.304 Read-Only File System**

2315 A file system that has implementation-defined characteristics restricting modifications.

2316 **Note:** File Times Update is described in detail in [Section 4.8](#) (on page 109).

2317 **3.305 Read-Write Lock**

2318 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous
 2319 read-only access to data while allowing only one thread to have write access at any given time.
 2320 They are typically used to protect data that is read-only more frequently than it is changed.

2321 Read-write locks can be used to synchronize threads in the current process and other processes if
 2322 they are allocated in memory that is writable and shared among the cooperating processes and
 2323 have been initialized for this behavior.

*Definitions**Real Group ID*2324 **3.306 Real Group ID**

2325 The attribute of a process that, at the time of process creation, identifies the group of the user
 2326 who created the process; see also [Section 3.188](#) (on page 63).

2327 **3.307 Real Time**

2328 Time measured as total units elapsed by the system clock without regard to which thread is
 2329 executing.

2330 **3.308 Realtime Signal Extension**

2331 A determinism improvement facility to enable asynchronous signal notifications to an
 2332 application to be queued without impacting compatibility with the existing signal functions.

2333 **3.309 Real User ID**

2334 The attribute of a process that, at the time of process creation, identifies the user who created the
 2335 process; see also [Section 3.428](#) (on page 102).

2336 **3.310 Record**

2337 A collection of related data units or words which is treated as a unit.

2338 **3.311 Redirection**

2339 In the shell command language, a method of associating files with the input or output of
 2340 commands.

2341 **Note:** For further information, see XCU [Section 2.7](#) (on page 2312).

2342 **3.312 Redirection Operator**

2343 In the shell command language, a token that performs a redirection function. It is one of the
 2344 following symbols:

2345 < > >| << >> <& >& <<- <>

2346 **3.313 Referenced Shared Memory Object**

2347 A shared memory object that is open or has one or more mappings defined on it.

2348 **3.314 Refresh**

2349 To ensure that the information on the user's terminal screen is up-to-date.

2350 **3.315 Regular Expression**

2351 A pattern that selects specific strings from a set of character strings.

2352 **Note:** Regular Expressions are described in detail in [Chapter 9](#) (on page 181).

2353 **3.316 Region**

2354 In the context of the address space of a process, a sequence of addresses.

2355 In the context of a file, a sequence of offsets.

2356 **3.317 Regular File**

2357 A file that is a randomly accessible sequence of bytes, with no further structure imposed by the
 2358 system.

Definitions

Relative Pathname

3.318 Relative Pathname

A pathname not beginning with a <slash> character.

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

3.319 Relocatable File

A file holding code or data suitable for linking with other object files to create an executable or a shared object file.

3.320 Relocation

The process of connecting symbolic references with symbolic definitions. For example, when a program calls a function, the associated call instruction transfers control to the proper destination address at execution.

3.321 Requested Batch Service

A service that is either rejected or performed prior to a response from the service to the requester.

3.322 (Time) Resolution

The minimum time interval that a clock can measure or whose passage a timer can detect.

3.323 Robust Mutex

A mutex with the *robust* attribute set.

Note: The *robust* attribute is defined in detail by the `pthread_mutexattr_getrobust()` function.

3.324 Root Directory

A directory, associated with a process, that is used in pathname resolution for pathnames that begin with a <slash> character.

3.325 Runnable Process (or Thread)

A thread that is capable of being a running thread, but for which no processor is available.

2382 3.326 Running Process (or Thread)

2383 A thread currently executing on a processor. On multi-processor systems there may be more
2384 than one such thread in a system at a time.

2385 3.327 Saved Resource Limits

2386 An attribute of a process that provides some flexibility in the handling of unrepresentable
2387 resource limits, as described in the *exec* family of functions and *setrlimit()*.

2388 **Note:** The *exec* and *setrlimit()* functions are defined in detail in the System Interfaces volume of
2389 POSIX.1-2008.

2390 3.328 Saved Set-Group-ID

2391 An attribute of a process that allows some flexibility in the assignment of the effective group ID
2392 attribute, as described in the *exec* family of functions and *setgid()*.

2393 **Note:** The *exec* and *setgid()* functions are defined in detail in the System Interfaces volume of
2394 POSIX.1-2008.

2395 3.329 Saved Set-User-ID

2396 An attribute of a process that allows some flexibility in the assignment of the effective user ID
2397 attribute, as described in the *exec* family of functions and *setuid()*.

2398 **Note:** The *exec* and *setuid()* functions are defined in detail in the System Interfaces volume of
2399 POSIX.1-2008.

2400 3.330 Scheduling

2401 The application of a policy to select a runnable process or thread to become a running process or
2402 thread, or to alter one or more of the thread lists.

2403 3.331 Scheduling Allocation Domain

2404 The set of processors on which an individual thread can be scheduled at any given time.

2405 3.332 Scheduling Contention Scope

2406 A property of a thread that defines the set of threads against which that thread competes for
2407 resources.

*Definitions**Scheduling Contention Scope*

2408 For example, in a scheduling decision, threads sharing scheduling contention scope compete for
 2409 processor resources. In POSIX.1-2008, a thread has scheduling contention scope of either
 2410 PTHREAD_SCOPE_SYSTEM or PTHREAD_SCOPE_PROCESS.

2411 3.333 Scheduling Policy

2412 A set of rules that is used to determine the order of execution of processes or threads to achieve
 2413 some goal.

2414 **Note:** Scheduling Policy is defined in detail in [Section 4.14](#) (on page 112).

2415 3.334 Screen

2416 A rectangular region of columns and lines on a terminal display. A screen may be a portion of a
 2417 physical display device or may occupy the entire physical area of the display device.

2418 3.335 Scroll

2419 To move the representation of data vertically or horizontally relative to the terminal screen.
 2420 There are two types of scrolling:

- 2421 1. The cursor moves with the data.
- 2422 2. The cursor remains stationary while the data moves.

2423 3.336 Semaphore

2424 A minimum synchronization primitive to serve as a basis for more complex synchronization
 2425 mechanisms to be defined by the application program.

2426 **Note:** Semaphores are defined in detail in [Section 4.16](#) (on page 113).

2427 **3.337 Session**

2428 A collection of process groups established for job control purposes. Each process group is a
 2429 member of a session. A process is considered to be a member of the session of which its process
 2430 group is a member. A newly created process joins the session of its creator. A process can alter
 2431 its session membership; see *setsid()*. There can be multiple process groups in the same session.

2432 **Note:** The *setsid()* function is defined in detail in the System Interfaces volume of POSIX.1-2008.

2433 **3.338 Session Leader**

2434 A process that has created a session.

2435 **Note:** For further information, see the *setsid()* function defined in the System Interfaces volume of
 2436 POSIX.1-2008.

2437 **3.339 Session Lifetime**

2438 The period between when a session is created and the end of the lifetime of all the process
 2439 groups that remain as members of the session.

2440 **3.340 Shared Memory Object**

2441 An object that represents memory that can be mapped concurrently into the address space of
 2442 more than one process.

2443 **3.341 Shell**

2444 A program that interprets sequences of text input as commands. It may operate on an input
 2445 stream or it may interactively prompt and read commands from a terminal.

2446 **3.342 Shell, the**

2447 The Shell Command Language Interpreter; a specific instance of a shell.

2448 **Note:** For further information, see the *sh* utility defined in the Shell and Utilities volume of
 2449 POSIX.1-2008.

3.343 Shell Script

A file containing shell commands. If the file is made executable, it can be executed by specifying its name as a simple command. Execution of a shell script causes a shell to execute the commands within the script. Alternatively, a shell can be requested to execute the commands in a shell script by specifying the name of the shell script as the operand to the *sh* utility.

Note: Simple Commands are defined in detail in XCU [Section 2.9.1](#) (on page 2316).

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

3.344 Signal

A mechanism by which a process or thread may be notified of, or affected by, an event occurring in the system. Examples of such events include hardware exceptions and specific actions by processes. The term signal is also used to refer to the event itself.

3.345 Signal Stack

Memory established for a thread, in which signal handlers catching signals sent to that thread are executed.

3.346 Single-Quote Character

The character designated by ' \ ' ' in the C language, also known as <apostrophe>.

3.347 Slash Character (<slash>)

The character ' / ', also known as solidus.

3.348 Socket

A file of a particular type that is used as a communications endpoint for process-to-process communication as described in the System Interfaces volume of POSIX.1-2008.

3.349 Socket Address

An address associated with a socket or remote endpoint, including an address family identifier and addressing information specific to that address family. The address may include multiple parts, such as a network address associated with a host system and an identifier for a specific endpoint.

2476 **3.350 Soft Limit**

2477 A resource limitation established for each process that the process may set to any value less than
 2478 or equal to the hard limit.

2479 **3.351 Source Code**

2480 When dealing with the Shell Command Language, input to the command language interpreter.
 2481 The term “shell script” is synonymous with this meaning.

2482 When dealing with an ISO/IEC-conforming programming language, source code is input to a
 2483 compiler conforming to that ISO/IEC standard.

2484 Source code also refers to the input statements prepared for the following standard utilities: *awk*,
 2485 *bc*, *ed*, *lex*, *localedef*, *make*, *sed*, and *yacc*.

2486 Source code can also refer to a collection of sources meeting any or all of these meanings.

2487 **Note:** The *awk*, *bc*, *ed*, *lex*, *localedef*, *make*, *sed*, and *yacc* utilities are defined in detail in the Shell and
 2488 Utilities volume of POSIX.1-2008.

2489 **3.352 Space Character (<space>)**

2490 The character defined in the portable character set as <space>. The <space> character is a
 2491 member of the **space** character class of the current locale, but represents the single character, and
 2492 not all of the possible members of the class; see also [Section 3.434](#) (on page 103).

2493 **3.353 Spawn**

2494 A process creation primitive useful for systems that have difficulty with *fork()* and as an efficient
 2495 replacement for *fork()/exec*.

2496 **3.354 Special Built-In**

2497 See *Built-In Utility* in [Section 3.83](#) (on page 46).

*Definitions**Special Parameter***3.355 Special Parameter**

In the shell command language, a parameter named by a single character from the following list:

* @ # ? ! - \$ 0

Note: For further information, see XCU [Section 2.5.2](#) (on page 2302).

3.356 Spin Lock

A synchronization object used to allow multiple threads to serialize their access to shared data.

3.357 Sporadic Server

A scheduling policy for threads and processes that reserves a certain amount of execution capacity for processing aperiodic events at a given priority level.

3.358 Standard Error

An output stream usually intended to be used for diagnostic messages.

3.359 Standard Input

An input stream usually intended to be used for primary data input.

3.360 Standard Output

An output stream usually intended to be used for primary data output.

3.361 Standard Utilities

The utilities described in the Shell and Utilities volume of POSIX.1-2008.

2515 **3.362 Stream**

2516 Appearing in lowercase, a stream is a file access object that allows access to an ordered sequence
 2517 of characters, as described by the ISO C standard. Such objects can be created by the *fdopen()*,
 2518 *fmemopen()*, *fopen()*, *open_memstream()*, or *popen()* functions, and are associated with a file
 2519 descriptor. A stream provides the additional services of user-selectable buffering and formatted
 2520 input and output; see also [Section 3.363](#).

2521 **Note:** For further information, see XSH [Section 2.5](#) (on page 490).

2522 The *fdopen()*, *fmemopen()*, *fopen()*, *open_memstream()*, and *popen()* functions are defined in detail
 2523 in the System Interfaces volume of POSIX.1-2008.

2524 **3.363 STREAM**

2525 Appearing in uppcase, STREAM refers to a full-duplex connection between a process and an
 2526 open device or pseudo-device. It optionally includes one or more intermediate processing
 2527 modules that are interposed between the process end of the STREAM and the device driver (or
 2528 pseudo-device driver) end of the STREAM; see also [Section 3.362](#).

2529 **Note:** For further information, see XSH [Section 2.6](#) (on page 494).

2530 **3.364 STREAM End**

2531 The STREAM end is the driver end of the STREAM and is also known as the downstream end of
 2532 the STREAM.

2533 **3.365 STREAM Head**

2534 The STREAM head is the beginning of the STREAM and is at the boundary between the system
 2535 and the application process. This is also known as the upstream end of the STREAM.

2536 **3.366 STREAMS Multiplexor**

2537 A driver with multiple STREAMS connected to it. Multiplexing with STREAMS connected
 2538 above is referred to as N-to-1, or “upper multiplexing”. Multiplexing with STREAMS connected
 2539 below is referred to as 1-to-N or “lower multiplexing”.

2540 **3.367 String**

2541 A contiguous sequence of bytes terminated by and including the first null byte.

3.368 Subshell

A shell execution environment, distinguished from the main or current shell execution environment.

Note: For further information, see XCU [Section 2.12](#) (on page 2331).

3.369 Successfully Transferred

For a write operation to a regular file, when the system ensures that all data written is readable on any subsequent open of the file (even one that follows a system or power failure) in the absence of a failure of the physical storage medium.

For a read operation, when an image of the data on the physical storage medium is available to the requesting process.

3.370 Supplementary Group ID

An attribute of a process used in determining file access permissions. A process has up to {NGROUPS_MAX} supplementary group IDs in addition to the effective group ID. The supplementary group IDs of a process are set to the supplementary group IDs of the parent process when the process is created.

3.371 Suspended Job

A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the process group to stop. A suspended job is a background job, but a background job is not necessarily a suspended job.

3.372 Symbolic Constant

An object-like macro defined with a constant value.

Unless stated otherwise, the following shall apply to every symbolic constant:

- It expands to a compile-time constant expression with an integer type.
- It may be defined as another type of constant—e.g., an enumeration constant—as well as being a macro.
- It need not be usable in **#if** preprocessing directives.

3.373 Symbolic Link

A type of file with the property that when the file is encountered during pathname resolution, a string stored by the file is used to modify the pathname resolution. The stored string has a length of {SYMLINK_MAX} bytes or fewer.

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

3.374 Synchronized Input and Output

A determinism and robustness improvement mechanism to enhance the data input and output mechanisms, so that an application can ensure that the data being manipulated is physically present on secondary mass storage devices.

3.375 Synchronized I/O Completion

The state of an I/O operation that has either been successfully transferred or diagnosed as unsuccessful.

3.376 Synchronized I/O Data Integrity Completion

For read, when the operation has been completed or diagnosed if unsuccessful. The read is complete only when an image of the data has been successfully transferred to the requesting process. If there were any pending write requests affecting the data to be read at the time that the synchronized read operation was requested, these write requests are successfully transferred prior to reading the data.

For write, when the operation has been completed or diagnosed if unsuccessful. The write is complete only when the data specified in the write request is successfully transferred and all file system information required to retrieve the data is successfully transferred.

File attributes that are not necessary for data retrieval (access time, modification time, status change time) need not be successfully transferred prior to returning to the calling process.

3.377 Synchronized I/O File Integrity Completion

Identical to a synchronized I/O data integrity completion with the addition that all file attributes relative to the I/O operation (including access time, modification time, status change time) are successfully transferred prior to returning to the calling process.

3.378 Synchronized I/O Operation

An I/O operation performed on a file that provides the application assurance of the integrity of its data and files.

Definitions

Synchronous I/O Operation

3.379 Synchronous I/O Operation

An I/O operation that causes the thread requesting the I/O to be blocked from further use of the processor until that I/O operation completes.

Note: A synchronous I/O operation does not imply synchronized I/O data integrity completion or synchronized I/O file integrity completion.

3.380 Synchronously-Generated Signal

A signal that is attributable to a specific thread.

For example, a thread executing an illegal instruction or touching invalid memory causes a synchronously-generated signal. Being synchronous is a property of how the signal was generated and not a property of the signal number.

3.381 System

An implementation of POSIX.1-2008.

3.382 System Boot

An unspecified sequence of events that may result in the loss of transitory data; that is, data that is not saved in permanent storage. For example, message queues, shared memory, semaphores, and processes.

3.383 System Clock

A clock with at least one second resolution that contains seconds since the Epoch.

3.384 System Console

A device that receives messages sent by the *syslog()* function, and the *fntmsg()* function when the MM_CONSOLE flag is set.

Note: The *syslog()* and *fntmsg()* functions are defined in detail in the System Interfaces volume of POSIX.1-2008.

3.385 System Crash

An interval initiated by an unspecified circumstance that causes all processes (possibly other than special system processes) to be terminated in an undefined manner, after which any

2624 changes to the state and contents of files created or written to by an application prior to the
2625 interval are undefined, except as required elsewhere in POSIX.1-2008.

2626 **3.386 System Databases**

2627 An implementation provides two system databases: the “group database” (see also [Section](#)
2628 [3.187](#), on page 63) and the “user database” (see also [Section 3.427](#), on page 101).

2629 **3.387 System Documentation**

2630 All documentation provided with an implementation except for the conformance document.
2631 Electronically distributed documents for an implementation are considered part of the system
2632 documentation.

2633 **3.388 System Process**

2634 An object other than a process executing an application, that is provided by the system and has a
2635 process ID.

2636 **3.389 System Reboot**

2637 See *System Boot* defined in [Section 3.382](#) (on page 95).

2638 **3.390 System Trace Event**

2639 A trace event that is generated by the implementation, in response either to a system-initiated
2640 action or to an application-requested action, except for a call to *posix_trace_event()*. When
2641 supported by the implementation, a system-initiated action generates a process-independent
2642 system trace event and an application-requested action generates a process-dependent system
2643 trace event. For a system trace event not defined by POSIX.1-2008, the associated trace event
2644 type identifier is derived from the implementation-defined name for this trace event, and the
2645 associated data is of implementation-defined content and length.

2646 **3.391 System-Wide**

2647 Pertaining to events occurring in all processes existing in an implementation at a given point in
2648 time.

Definitions

Tab Character (<tab>)

3.392 Tab Character (<tab>)

A character that in the output stream indicates that printing or displaying should start at the next horizontal tabulation position on the current line. It is the character designated by '`\t`' in the C language. If the current position is at or past the last defined horizontal tabulation position, the behavior is unspecified. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the tabulation.

3.393 Terminal (or Terminal Device)

A character special file that obeys the specifications of the general terminal interface.

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

3.394 Text Column

A roughly rectangular block of characters capable of being laid out side-by-side next to other text columns on an output page or terminal screen. The widths of text columns are measured in column positions.

3.395 Text File

A file that contains characters organized into zero or more lines. The lines do not contain NUL characters and none can exceed `{LINE_MAX}` bytes in length, including the `<newline>` character. Although POSIX.1-2008 does not distinguish between text files and binary files (see the ISO C standard), many utilities only produce predictable or meaningful output when operating on text files. The standard utilities that have such restrictions always specify "text files" in their STDIN or INPUT FILES sections.

3.396 Thread

A single flow of control within a process. Each thread has its own thread ID, scheduling priority and policy, *errno* value, thread-specific key/value bindings, and the required system resources to support a flow of control. Anything whose address may be determined by a thread, including but not limited to static variables, storage obtained via `malloc()`, directly addressable storage obtained through implementation-defined functions, and automatic variables, are accessible to all threads in the same process.

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

3.397 Thread ID

Each thread in a process is uniquely identified during its lifetime by a value of type `pthread_t` called a thread ID.

2680 **3.398 Thread List**

2681 An ordered set of runnable threads that all have the same ordinal value for their priority.

2682 The ordering of threads on the list is determined by a scheduling policy or policies. The set of
2683 thread lists includes all runnable threads in the system.2684 **3.399 Thread-Safe**2685 A function that may be safely invoked concurrently by multiple threads. Each function defined
2686 in the System Interfaces volume of POSIX.1-2008 is thread-safe unless explicitly stated
2687 otherwise. Examples are any “pure” function, a function which holds a mutex locked while it is
2688 accessing static storage, or objects shared among threads.2689 **3.400 Thread-Specific Data Key**2690 A process global handle of type `pthread_key_t` which is used for naming thread-specific data.2691 Although the same key value may be used by different threads, the values bound to the key by
2692 `pthread_setspecific()` and accessed by `pthread_getspecific()` are maintained on a per-thread basis
2693 and persist for the life of the calling thread.2694 **Note:** The `pthread_getspecific()` and `pthread_setspecific()` functions are defined in detail in the System
2695 Interfaces volume of POSIX.1-2008.2696 **3.401 Tilde Character (<tilde>)**2697 The character `'~'`.2698 **3.402 Timeouts**2699 A method of limiting the length of time an interface will block; see also [Section 3.76](#) (on page 44).2700 **3.403 Timer**2701 A mechanism that can notify a thread when the time as measured by a particular clock has
2702 reached or passed a specified value, or when a specified amount of time has passed.2703 **3.404 Timer Overrun**2704 A condition that occurs each time a timer, for which there is already an expiration signal queued
2705 to the process, expires.

*Definitions**Token*2706 **3.405 Token**

2707 In the shell command language, a sequence of characters that the shell considers as a single unit
 2708 when reading input. A token is either an operator or a word.

2709 **Note:** The rules for reading input are defined in detail in XCU [Section 2.3](#) (on page 2299).

2710 **3.406 Trace Analyzer Process**

2711 A process that extracts trace events from a trace stream to retrieve information about the
 2712 behavior of an application.

2713 **3.407 Trace Controller Process**

2714 A process that creates a trace stream for tracing a process.

2715 **3.408 Trace Event**

2716 A data object that represents an action executed by the system, and that is recorded in a trace
 2717 stream.

2718 **3.409 Trace Event Type**

2719 A data object type that defines a class of trace event.

2720 **3.410 Trace Event Type Mapping**

2721 A one-to-one mapping between trace event types and trace event names.

2722 **3.411 Trace Filter**

2723 A filter that allows the trace controller process to specify those trace event types that are to be
 2724 ignored; that is, not generated.

2725 **3.412 Trace Generation Version**

2726 A data object that is an implementation-defined character string, generated by the trace system
 2727 and describing the origin and version of the trace system.

2728 3.413 Trace Log

2729 The flushed image of a trace stream, if the trace stream is created with a trace log.

2730 3.414 Trace Point

2731 An action that may cause a trace event to be generated.

2732 3.415 Trace Stream

2733 An opaque object that contains trace events plus internal data needed to interpret those trace
2734 events.

2735 3.416 Trace Stream Identifier

2736 A handle to manage tracing operations in a trace stream.

2737 3.417 Trace System

2738 A system that allows both system and user trace events to be generated into a trace stream.
2739 These trace events can be retrieved later.

2740 3.418 Traced Process

2741 A process for which at least one trace stream has been created. A traced process is also called a
2742 target process.

2743 3.419 Tracing Status of a Trace Stream

2744 A status that describes the state of an active trace stream. The tracing status of a trace stream can
2745 be retrieved from the trace stream attributes. An active trace stream can be in one of two states:
2746 running or suspended.

2747 3.420 Typed Memory Name Space

2748 A system-wide name space that contains the names of the typed memory objects present in the
2749 system. It is configurable for a given implementation.

Definitions

Typed Memory Object

3.421 Typed Memory Object

A combination of a typed memory pool and a typed memory port. The entire contents of the pool are accessible from the port. The typed memory object is identified through a name that belongs to the typed memory name space.

3.422 Typed Memory Pool

An extent of memory with the same operational characteristics. Typed memory pools may be contained within each other.

3.423 Typed Memory Port

A hardware access path to one or more typed memory pools.

3.424 Unbind

Remove the association between a network address and an endpoint.

3.425 Unit Data

See *Datagram* in [Section 3.124](#) (on page 53).

3.426 Upshifting

The conversion of a lowercase character that has a single-character uppercase representation into this uppercase representation.

3.427 User Database

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

- User name
- Numerical user ID
- Initial numerical group ID
- Initial working directory
- Initial user program

The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under which the initial values are operative are implementation-defined.

- 2775 If the initial user program field is null, an implementation-defined program is used.
- 2776 If the initial working directory field is null, the interpretation of that field is implementation-defined.
- 2777
- 2778 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2008.

2779 3.428 User ID

- 2780 A non-negative integer that is used to identify a system user. When the identity of a user is associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or a saved set-user-ID.
- 2781
- 2782

2783 3.429 User Name

- 2784 A string that is used to identify a user; see also [Section 3.427](#) (on page 101). To be portable across systems conforming to POSIX.1-2008, the value is composed of characters from the portable filename character set. The <hyphen> character should not be used as the first character of a portable user name.
- 2785
- 2786
- 2787

2788 3.430 User Trace Event

- 2789 A trace event that is generated explicitly by the application as a result of a call to *posix_trace_event()*.
- 2790

2791 3.431 Utility

- 2792 A program, excluding special built-in utilities provided as part of the Shell Command Language, that can be called by name from a shell to perform a specific task, or related set of tasks.
- 2793
- 2794 **Note:** For further information on special built-in utilities, see XCU [Section 2.14](#) (on page 2334).

Definitions

Variable

3.432 Variable

In the shell command language, a named parameter.

Note: For further information, see XCU [Section 2.5](#) (on page 2301).

3.433 Vertical-Tab Character (<vertical-tab>)

A character that in the output stream indicates that printing should start at the next vertical tabulation position. It is the character designated by '`\v`' in the C language. If the current position is at or past the last defined vertical tabulation position, the behavior is unspecified. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the tabulation.

3.434 White Space

A sequence of one or more characters that belong to the `space` character class as defined via the `LC_CTYPE` category in the current locale.

In the POSIX locale, white space consists of one or more `<blank>` (`<space>` and `<tab>` characters), `<newline>`, `<carriage-return>`, `<form-feed>`, and `<vertical-tab>` characters.

3.435 Wide-Character Code (C Language)

An integer value corresponding to a single graphic symbol or control code.

Note: C Language Wide-Character Codes are defined in detail in [Section 6.3](#) (on page 129).

3.436 Wide-Character Input/Output Functions

The functions that perform wide-oriented input from streams or wide-oriented output to streams: `fgetwc()`, `fgetws()`, `fputwc()`, `fputws()`, `fwprintf()`, `fwscanf()`, `getwc()`, `getwchar()`, `putwc()`, `putwchar()`, `ungetwc()`, `vfwprintf()`, `vfwscanf()`, `vwprintf()`, `vwscanf()`, `wprintf()`, and `wscanf()`.

Note: These functions are defined in detail in the System Interfaces volume of POSIX.1-2008.

3.437 Wide-Character String

A contiguous sequence of wide-character codes terminated by and including the first null wide-character code.

2820 **3.438 Word**

2821 In the shell command language, a token other than an operator. In some cases a word is also a
 2822 portion of a word token: in the various forms of parameter expansion, such as $\${name-word}$,
 2823 and variable assignment, such as $name=word$, the word is the portion of the token depicted by
 2824 *word*. The concept of a word is no longer applicable following word expansions—only fields
 2825 remain.

2826 **Note:** For further information, see XCU [Section 2.6.2](#) (on page 2306) and [Section 2.6](#) (on page 2305).

2827 **3.439 Working Directory (or Current Working Directory)**

2828 A directory, associated with a process, that is used in pathname resolution for pathnames that do
 2829 not begin with a <slash> character.

2830 **3.440 Worldwide Portability Interface**

2831 Functions for handling characters in a codeset-independent manner.

2832 **3.441 Write**

2833 To output characters to a file, such as standard output or standard error. Unless otherwise stated,
 2834 standard output is the default output destination for all uses of the term “write”; see the
 2835 distinction between display and write in [Section 3.133](#) (on page 54).

2836 **3.442 XSI**

2837 The X/Open System Interfaces (XSI) option is the core application programming interface for C
 2838 and *sh* programming for systems conforming to the Single UNIX Specification. This is a
 2839 superset of the mandatory requirements for conformance to POSIX.1-2008.

*Definitions**XSI-Conformant*2840 **3.443 XSI-Conformant**

2841 A system which allows an application to be built using a set of services that are consistent across
2842 all systems that conform to POSIX.1-2008 and that support the XSI option.

2843 **Note:** See also [Chapter 2](#) (on page 15).

2844 **3.444 Zombie Process**

2845 A process that has terminated and that is deleted when its exit status has been reported to
2846 another process which is waiting for that process to terminate.

2847 **3.445 ± 0**

2848 The algebraic sign provides additional information about any variable that has the value zero
2849 when the representation allows the sign to be determined.

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

(blank page)

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

General Concepts

For the purposes of POSIX.1-2008, the general concepts given in [Chapter 4](#) apply.

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

4.1 Concurrent Execution

Functions that suspend the execution of the calling thread shall not cause the execution of other threads to be indefinitely suspended.

4.2 Directory Protection

If a directory is writable and the mode bit `S_ISVTX` is set on the directory, a process may remove or rename files within that directory only if one or more of the following is true:

- The effective user ID of the process is the same as that of the owner ID of the file.
- The effective user ID of the process is the same as that of the owner ID of the directory.
- The process has appropriate privileges.
- Optionally, the file is writable by the process. Whether or not files that are writable by the process can be removed or renamed is implementation-defined.

If the `S_ISVTX` bit is set on a non-directory file, the behavior is unspecified.

4.3 Extended Security Controls

An implementation may provide implementation-defined extended security controls (see [Section 3.160](#), on page 58). These permit an implementation to provide security mechanisms to implement different security policies than those described in POSIX.1-2008. These mechanisms shall not alter or override the defined semantics of any of the interfaces in POSIX.1-2008.

4.4 File Access Permissions

The standard file access control mechanism uses the file permission bits, as described below.

Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An additional access control mechanism shall only further restrict the access permissions defined by the file permission bits. An alternate file access control mechanism shall:

- Specify file permission bits for the file owner class, file group class, and file other class of that file, corresponding to the access permissions.
- Be enabled only by explicit user action, on a per-file basis by the file owner or a user with appropriate privileges.
- Be disabled for a file after the file permission bits are changed for that file with *chmod()*. The disabling of the alternate mechanism need not disable any additional mechanisms supported by an implementation.

Whenever a process requests file access permission for read, write, or execute/search, if no additional mechanism denies access, access shall be determined as follows:

- If a process has appropriate privileges:
 - If read, write, or directory search permission is requested, access shall be granted.
 - If execute permission is requested, access shall be granted if execute permission is granted to at least one user by the file permission bits or by an alternate access control mechanism; otherwise, access shall be denied.
- Otherwise:
 - The file permission bits of a file contain read, write, and execute/search permissions for the file owner class, file group class, and file other class.
 - Access shall be granted if an alternate access control mechanism is not enabled and the requested access permission bit is set for the class (file owner class, file group class, or file other class) to which the process belongs, or if an alternate access control mechanism is enabled and it allows the requested access; otherwise, access shall be denied.

POSIX.1-2008 does not provide a way to open a directory for searching. It is unspecified whether directory search permission is granted based on the file access modes of the directory's file descriptor or on the mode of the directory at the time the directory is searched.

4.5 File Hierarchy

Files in the system are organized in a hierarchical structure in which all of the non-terminal nodes are directories and all of the terminal nodes are any other type of file. Since multiple directory entries may refer to the same file, the hierarchy is properly described as a “directed graph”.

4.6 Filenames

Uppercase and lowercase letters shall retain their unique identities between conforming implementations.

4.7 Filename Portability

For a filename to be portable across implementations conforming to POSIX.1-2008, it shall consist only of the portable filename character set as defined in [Section 3.276](#) (on page 77).

Portable filenames shall not have the <hyphen> character as the first character since this may cause problems when filenames are passed as command line arguments.

4.8 File Times Update

Each file has three distinct associated timestamps: the time of last data access, the time of last data modification, and the time the file status last changed. These values are returned in the file characteristics structure **struct stat**, as described in [<sys/stat.h>](#) (on page 388).

Each function or utility in POSIX.1-2008 that reads or writes data (even if the data does not change) or performs an operation to change file status (even if the file status does not change) indicates which of the appropriate timestamps shall be marked for update. If an implementation of such a function or utility marks for update one of these timestamps in a place or time not specified by POSIX.1-2008, this shall be documented, except that any changes caused by pathname resolution need not be documented. For the other functions or utilities in POSIX.1-2008 (those that are not explicitly required to read or write file data or change file status, but that in some implementations happen to do so), the effect is unspecified.

An implementation may update timestamps that are marked for update immediately, or it may update such timestamps periodically. At the point in time when an update occurs, any marked timestamps shall be set to the current time and the update marks shall be cleared. All timestamps that are marked for update shall be updated when the file ceases to be open by any process or before a *fstat()*, *fstatat()*, *fsync()*, *futimens()*, *lstat()*, *stat()*, *utime()*, *utimensat()*, or *utimes()* is successfully performed on the file. Other times at which updates are done are unspecified. Marks for update, and updates themselves, shall not be done for files on read-only file systems; see [Section 3.304](#) (on page 82).

The resolution of timestamps of files in a file system is implementation-defined, but shall be no coarser than one-second resolution. The three timestamps shall always have values that are supported by the file system. Whenever any of a file's timestamps are to be set to a value *V* according to the rules of the preceding paragraphs of this section, the implementation shall immediately set the timestamp to the greatest value supported by the file system that is not greater than *V*.

4.9 Host and Network Byte Orders

When data is transmitted over the network, it is sent as a sequence of octets (8-bit unsigned values). If an entity (such as an address or a port number) can be larger than 8 bits, it needs to be stored in several octets. The convention is that all such values are stored with 8 bits in each octet, and with the first (lowest-addressed) octet holding the most-significant bits. This is called “network byte order”.

Network byte order may not be convenient for processing actual values. For this, it is more sensible for values to be stored as ordinary integers. This is known as “host byte order”. In host byte order:

- The most significant bit might not be stored in the first byte in address order.
- Bits might not be allocated to bytes in any obvious order at all.

8-bit values stored in `uint8_t` objects do not require conversion to or from host byte order, as they have the same representation. 16 and 32-bit values can be converted using the `htonl()`, `htons()`, `ntohl()`, and `ntohs()` functions. When reading data that is to be converted to host byte order, it should either be received directly into a `uint16_t` or `uint32_t` object or should be copied from an array of bytes using `memcpy()` or similar. Passing the data through other types could cause the byte order to be changed. Similar considerations apply when sending data.

4.10 Measurement of Execution Time

The mechanism used to measure execution time shall be implementation-defined. The implementation shall also define to whom the CPU time that is consumed by interrupt handlers and system services on behalf of the operating system will be charged. See [Section 3.118](#) (on page 52).

4.11 Memory Synchronization

Applications shall ensure that access to any memory location by more than one thread of control (threads or processes) is restricted such that no thread of control can read or modify a memory location while another thread of control may be modifying it. Such access is restricted using functions that synchronize thread execution and also synchronize memory with respect to other threads. The following functions synchronize memory with respect to other threads:

<code>fork()</code>	<code>pthread_mutex_trylock()</code>	<code>pthread_rwlock_unlock()</code>
<code>pthread_barrier_wait()</code>	<code>pthread_mutex_unlock()</code>	<code>pthread_rwlock_wrlock()</code>
<code>pthread_cond_broadcast()</code>	<code>pthread_spin_lock()</code>	<code>sem_post()</code>
<code>pthread_cond_signal()</code>	<code>pthread_spin_trylock()</code>	<code>sem_timedwait()</code>
<code>pthread_cond_timedwait()</code>	<code>pthread_spin_unlock()</code>	<code>sem_trywait()</code>
<code>pthread_cond_wait()</code>	<code>pthread_rwlock_rdlock()</code>	<code>sem_wait()</code>
<code>pthread_create()</code>	<code>pthread_rwlock_timedrdlock()</code>	<code>semctl()</code>
<code>pthread_join()</code>	<code>pthread_rwlock_timedwrlock()</code>	<code>semop()</code>
<code>pthread_mutex_lock()</code>	<code>pthread_rwlock_tryrdlock()</code>	<code>wait()</code>
<code>pthread_mutex_timedlock()</code>	<code>pthread_rwlock_trywrlock()</code>	<code>waitpid()</code>

The `pthread_once()` function shall synchronize memory for the first call in each thread for a given `pthread_once_t` object.

The `pthread_mutex_lock()` function need not synchronize memory if the mutex type is `PTHREAD_MUTEX_RECURSIVE` and the calling thread already owns the mutex. The `pthread_mutex_unlock()` function need not synchronize memory if the mutex type is `PTHREAD_MUTEX_RECURSIVE` and the mutex has a lock count greater than one.

Unless explicitly stated otherwise, if one of the above functions returns an error, it is unspecified whether the invocation causes memory to be synchronized.

Applications may allow more than one thread of control to read a memory location simultaneously.

4.12 Pathname Resolution

Pathname resolution is performed for a process to resolve a pathname to a particular directory entry for a file in the file hierarchy. There may be multiple pathnames that resolve to the same directory entry, and multiple directory entries for the same file. When a process resolves a pathname of an existing directory entry, the entire pathname shall be resolved as described below. When a process resolves a pathname of a directory entry that is to be created immediately after the pathname is resolved, pathname resolution terminates when all components of the path prefix of the last component have been resolved. It is then the responsibility of the process to create the final component.

Each filename in the pathname is located in the directory specified by its predecessor (for example, in the pathname fragment `a/b`, file `b` is located in directory `a`). Pathname resolution shall fail if this cannot be accomplished. If the pathname begins with a `<slash>`, the predecessor of the first filename in the pathname shall be taken to be the root directory of the process (such pathnames are referred to as “absolute pathnames”). If the pathname does not begin with a `<slash>`, the predecessor of the first filename of the pathname shall be taken to be either the current working directory of the process or for certain interfaces the directory identified by a file descriptor passed to the interface (such pathnames are referred to as “relative pathnames”).

The interpretation of a pathname component is dependent on the value of `{NAME_MAX}` and `_POSIX_NO_TRUNC` associated with the path prefix of that component. If any pathname component is longer than `{NAME_MAX}`, the implementation shall consider this an error.

A pathname that contains at least one non-`<slash>` character and that ends with one or more trailing `<slash>` characters shall not be resolved successfully unless the last pathname component before the trailing `<slash>` characters names an existing directory or a directory entry that is to be created for a directory immediately after the pathname is resolved. Interfaces using pathname resolution may specify additional constraints⁶ when a pathname that does not name an existing directory contains at least one non-`<slash>` character and contains one or more trailing `<slash>` characters.

If a symbolic link is encountered during pathname resolution, the behavior shall depend on whether the pathname component is at the end of the pathname and on the function being performed. If all of the following are true, then pathname resolution is complete:

1. This is the last pathname component of the pathname.
2. The pathname has no trailing `<slash>`.

6. The only interfaces that further constrain pathnames in POSIX.1-2008 are the `rename()` and `renameat()` functions (see XSH `rename()`) and the `mv` utility (see XCU `mv`).

3. The function is required to act on the symbolic link itself, or certain arguments direct that the function act on the symbolic link itself.

In all other cases, the system shall prefix the remaining pathname, if any, with the contents of the symbolic link. If the combined length exceeds {PATH_MAX}, and the implementation considers this to be an error, *errno* shall be set to [ENAMETOOLONG] and an error indication shall be returned. Otherwise, the resolved pathname shall be the resolution of the pathname just created. If the resulting pathname does not begin with a <slash>, the predecessor of the first filename of the pathname is taken to be the directory containing the symbolic link.

If the system detects a loop in the pathname resolution process, it shall set *errno* to [ELOOP] and return an error indication. The same may happen if during the resolution process more symbolic links were followed than the implementation allows. This implementation-defined limit shall not be smaller than {SYMLOOP_MAX}.

The special filename dot shall refer to the directory specified by its predecessor. The special filename dot-dot shall refer to the parent directory of its predecessor directory. As a special case, in the root directory, dot-dot may refer to the root directory itself.

A pathname consisting of a single <slash> shall resolve to the root directory of the process. A null pathname shall not be successfully resolved. A pathname that begins with two successive <slash> characters may be interpreted in an implementation-defined manner, although more than two leading <slash> characters shall be treated as a single <slash> character.

Pathname resolution for a given pathname shall yield the same results when used by any interface in POSIX.1-2008 as long as there are no changes to any files evaluated during pathname resolution for the given pathname between resolutions.

4.13 Process ID Reuse

A process group ID shall not be reused by the system until the process group lifetime ends.

A process ID shall not be reused by the system until the process lifetime ends. In addition, if there exists a process group whose process group ID is equal to that process ID, the process ID shall not be reused by the system until the process group lifetime ends. A process that is not a system process shall not have a process ID of 1.

4.14 Scheduling Policy

A scheduling policy affects process or thread ordering:

- When a process or thread is a running thread and it becomes a blocked thread
- When a process or thread is a running thread and it becomes a preempted thread
- When a process or thread is a blocked thread and it becomes a runnable thread
- When a running thread calls a function that can change the priority or scheduling policy of a process or thread
- In other scheduling policy-defined circumstances

Conforming implementations shall define the manner in which each of the scheduling policies may modify the priorities or otherwise affect the ordering of processes or threads at each of the occurrences listed above. Additionally, conforming implementations shall define in what other

circumstances and in what manner each scheduling policy may modify the priorities or affect the ordering of processes or threads.

4.15 Seconds Since the Epoch

A value that approximates the number of seconds that have elapsed since the Epoch. A Coordinated Universal Time name (specified in terms of seconds (*tm_sec*), minutes (*tm_min*), hours (*tm_hour*), days since January 1 of the year (*tm_yday*), and calendar year minus 1900 (*tm_year*)) is related to a time represented as seconds since the Epoch, according to the expression below.

If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and the value is non-negative, the value is related to a Coordinated Universal Time name according to the C-language expression, where *tm_sec*, *tm_min*, *tm_hour*, *tm_yday*, and *tm_year* are all integer types:

$$tm\_sec + tm\_min*60 + tm\_hour*3600 + tm\_yday*86400 + \\ (tm\_year-70)*31536000 + ((tm\_year-69)/4)*86400 - \\ ((tm\_year-1)/100)*86400 + ((tm\_year+299)/400)*86400$$

The relationship between the actual time of day and the current value for seconds since the Epoch is unspecified.

How any changes to the value of seconds since the Epoch are made to align to a desired relationship with the current actual time is implementation-defined. As represented in seconds since the Epoch, each and every day shall be accounted for by exactly 86 400 seconds.

Note: The last three terms of the expression add in a day for each year that follows a leap year starting with the first leap year since the Epoch. The first term adds a day every 4 years starting in 1973, the second subtracts a day back out every 100 years starting in 2001, and the third adds a day back in every 400 years starting in 2001. The divisions in the formula are integer divisions; that is, the remainder is discarded leaving only the integer quotient.

4.16 Semaphore

A minimum synchronization primitive to serve as a basis for more complex synchronization mechanisms to be defined by the application program.

For the semaphores associated with the Semaphores option, a semaphore is represented as a shareable resource that has a non-negative integer value. When the value is zero, there is a (possibly empty) set of threads awaiting the availability of the semaphore.

For the semaphores associated with the X/Open System Interfaces (XSI) option, a semaphore is a positive integer (0 through 32767). The *semget()* function can be called to create a set or array of semaphores. A semaphore set can contain one or more semaphores up to an implementation-defined value.

Semaphore Lock Operation

An operation that is applied to a semaphore. If, prior to the operation, the value of the semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and added to the set of threads awaiting the semaphore; otherwise, the value shall be decremented.

Semaphore Unlock Operation

An operation that is applied to a semaphore. If, prior to the operation, there are any threads in the set of threads awaiting the semaphore, then some thread from that set shall be removed from the set and becomes unblocked; otherwise, the semaphore value shall be incremented.

4.17 Thread-Safety

Refer to XSH [Section 2.9](#) (on page 507).

4.18 Tracing

The trace system allows a traced process to have a selection of events created for it. Traces consist of streams of trace event types.

A trace event type is identified on the one hand by a trace event type name, also referenced as a trace event name, and on the other hand by a trace event type identifier. A trace event name is a human-readable string. A trace event type identifier is an opaque identifier used by the trace system. There shall be a one-to-one relationship between trace event type identifiers and trace event names for a given trace stream and also for a given traced process. The trace event type identifier shall be generated automatically from a trace event name by the trace system either when a trace controller process invokes `posix_trace_trid_eventid_open()` or when an instrumented application process invokes `posix_trace_eventid_open()`. Trace event type identifiers are used to filter trace event types, to allow interpretation of user data, and to identify the kind of trace point that generated a trace event.

Each trace event shall be of a particular trace event type, and associated with a trace event type identifier. The execution of a trace point shall generate a trace event if a trace stream has been created and started for the process that executed the trace point and if the corresponding trace event type identifier is not ignored by filtering.

A generated trace event shall be recorded in a trace stream, and optionally also in a trace log if a trace log is associated with the trace stream, except that:

- For a trace stream, if no resources are available for the event, the event is lost.
- For a trace log, if no resources are available for the event, or a flush operation does not succeed, the event is lost.

A trace event recorded in an active trace stream may be retrieved by an application having appropriate privileges.

A trace event recorded in a trace log may be retrieved by an application having appropriate privileges after opening the trace log as a pre-recorded trace stream, with the function `posix_trace_open()`.

When a trace event is reported it is possible to retrieve the following:

- A trace event type identifier
- A timestamp
- The process ID of the traced process, if the trace event is process-dependent
- Any optional trace event data including its length
- If the Threads option is supported, the thread ID, if the trace event is process-dependent
- The program address at which the trace point was invoked

Trace events may be mapped from trace event types to trace event names. One such mapping shall be associated with each trace stream. An active trace stream is associated with a traced process, and also with its children if the Trace Inherit option is supported and also the inheritance policy is set to `_POSIX_TRACE_INHERIT`. Therefore each traced process has a mapping of the trace event names to trace event type identifiers that have been defined for that process.

Traces can be recorded into either trace streams or trace logs.

The implementation and format of a trace stream are unspecified. A trace stream need not be and generally is not persistent. A trace stream may be either active or pre-recorded:

- An active trace stream is a trace stream that has been created and has not yet been shut down. It can be of one of the two following classes:
 1. An active trace stream without a trace log that was created with the `posix_trace_create()` function
 2. If the Trace Log option is supported, an active trace stream with a trace log that was created with the `posix_trace_create_withlog()` function
- A pre-recorded trace stream is a trace stream that was opened from a trace log object using the `posix_trace_open()` function.

An active trace stream can loop. This behavior means that when the resources allocated by the trace system for the trace stream are exhausted, the trace system reuses the resources associated with the oldest recorded trace events to record new trace events.

If the Trace Log option is supported, an active trace stream with a trace log can be flushed. This operation causes the trace system to write trace events from the trace stream to the associated trace log, following the defined policies or using an explicit function call. After this operation, the trace system may reuse the resources associated with the flushed trace events.

An active trace stream with or without a trace log can be cleared. This operation shall cause all the resources associated with this trace stream to be reinitialized. The trace stream shall behave as if it was returning from its creation, except that the mapping of trace event type identifiers to trace event names shall not be cleared. If a trace log was associated with this trace stream, the trace log shall also be reinitialized.

A trace log shall be recorded when the `posix_trace_shutdown()` operation is invoked or during tracing, depending on the tracing strategy which is defined by a log policy. After the trace stream has been shut down, the trace information can be retrieved from the associated trace log using the same interface used to retrieve information from an active trace stream.

For a traced process, if the Trace Inherit option is supported and the trace stream's inheritance attribute is `_POSIX_TRACE_INHERIT`, the initial targeted traced process shall be traced together with all of its future children. The `posix_pid` member of each trace event in a trace stream shall be the process ID of the traced process.

Each trace point may be an implementation-defined action such as a context switch, or an application-programmed action such as a call to a specific operating system service (for example, `fork()`) or a call to `posix_trace_event()`.

Trace points may be filtered. The operation of the filter is to filter out (ignore) selected trace events. By default, no trace events are filtered.

The results of the tracing operations can be analyzed and monitored by a trace controller process or a trace analyzer process.

Only the trace controller process has control of the trace stream it has created. The control of the operation of a trace stream is done using its corresponding trace stream identifier. The trace controller process is able to:

- Initialize the attributes of a trace stream
- Create the trace stream
- Start and stop tracing
- Know the mapping of the traced process
- If the Trace Event Filter option is supported, filter the type of trace events to be recorded
- Shut the trace stream down

A traced process may also be a trace controller process. Only the trace controller process can control its trace stream(s). A trace stream created by a trace controller process shall be shut down if its controller process terminates or executes another file.

A trace controller process may also be a trace analyzer process. Trace analysis can be done concurrently with the traced process or can be done off-line, in the same or in a different platform.

4.19 Treatment of Error Conditions for Mathematical Functions

For all the functions in the `<math.h>` header, an application wishing to check for error situations should set `errno` to 0, and call `feclearexcept(FE_ALL_EXCEPT)` before calling the function. On return, if `errno` is non-zero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an error has occurred.

The following error conditions are defined for all functions in the `<math.h>` header.

4.19.1 Domain Error

A “domain error” shall occur if an input argument is outside the domain over which the mathematical function is defined. The description of each function lists any required domain errors; an implementation may define additional domain errors, provided that such errors are consistent with the mathematical definition of the function.

On a domain error, the function shall return an implementation-defined value; if the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero, `errno` shall be set to `[EDOM]`; if the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-zero, the “invalid” floating-point exception shall be raised.

4.19.2 Pole Error

A “pole error” occurs if the mathematical result of the function is an exact infinity (for example, $\log(0.0)$).

On a pole error, the function shall return the value of the macro `HUGE_VAL`, `HUGE_VALF`, or `HUGE_VALL` according to the return type, with the same sign as the correct value of the function; if the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero, `errno` shall be set to `[ERANGE]`; if the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-zero, the “divide-by-zero” floating-point exception shall be raised.

4.19.3 Range Error

A “range error” shall occur if the finite mathematical result of the function cannot be represented in an object of the specified type, due to extreme magnitude.

4.19.3.1 Result Overflows

A floating result overflows if the magnitude of the mathematical result is finite but so large that the mathematical result cannot be represented without extraordinary roundoff error in an object of the specified type. If a floating result overflows and default rounding is in effect, then the function shall return the value of the macro `HUGE_VAL`, `HUGE_VALF`, or `HUGE_VALL` according to the return type, with the same sign as the correct value of the function; if the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero, `errno` shall be set to `[ERANGE]`; if the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-zero, the “overflow” floating-point exception shall be raised.

4.19.3.2 Result Underflows

The result underflows if the magnitude of the mathematical result is so small that the mathematical result cannot be represented, without extraordinary roundoff error, in an object of the specified type. If the result underflows, the function shall return an implementation-defined value whose magnitude is no greater than the smallest normalized positive number in the specified type; if the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero, whether `errno` is set to `[ERANGE]` is implementation-defined; if the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-zero, whether the “underflow” floating-point exception is raised is implementation-defined.

4.20 Treatment of NaN Arguments for the Mathematical Functions

For functions called with a NaN argument, no errors shall occur and a NaN shall be returned, except where stated otherwise.

If a function with one or more NaN arguments returns a NaN result, the result should be the same as one of the NaN arguments (after possible type conversion), except perhaps for the sign.

On implementations that support the IEC 60559:1989 standard floating point, functions with signaling NaN argument(s) shall be treated as if the function were called with an argument that is a required domain error and shall return a quiet NaN result, except where stated otherwise.

Note: The function might never see the signaling NaN, since it might trigger when the arguments are evaluated during the function call.

On implementations that support the IEC 60559:1989 standard floating point, for those functions that do not have a documented domain error, the following shall apply:

These functions shall fail if:

Domain Error Any argument is a signaling NaN.

Either, the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero and *errno* shall be set to [EDOM], or the integer expression (`math_errhandling & MATH_ERREXCEPT`) is non-zero and the invalid floating-point exception shall be raised.

4.21 Utility

A utility program shall be either an executable file, such as might be produced by a compiler or linker system from computer source code, or a file of shell source code, directly interpreted by the shell. The program may have been produced by the user, provided by the system implementor, or acquired from an independent distributor.

The system may implement certain utilities as shell functions (see XCU Section 2.9.5, on page 2324) or built-in utilities, but only an application that is aware of the command search order (as described in XCU Section 2.9.1.1, on page 2317) or of performance characteristics can discern differences between the behavior of such a function or built-in utility and that of an executable file.

4.22 Variable Assignment

In the shell command language, a word consisting of the following parts:

varname=*value*

When used in a context where assignment is defined to occur and at no other time, the *value* (representing a word or field) shall be assigned as the value of the variable denoted by *varname*.

Note: For further information, see XCU Section 2.9.1 (on page 2316).

The *varname* and *value* parts shall meet the requirements for a name and a word, respectively, except that they are delimited by the embedded unquoted <equals-sign>, in addition to other delimiters.

General Concepts

Variable Assignment

- 3281 **Note:** Additional delimiters are described in XCU [Section 2.3](#) (on page 2299).
- 3282 When a variable assignment is done, the variable shall be created if it did not already exist. If
- 3283 *value* is not specified, the variable shall be given a null value.
- 3284 **Note:** An alternative form of variable assignment:
- 3285 *symbol=value*
- 3286 (where *symbol* is a valid word delimited by an <equals-sign>, but not a valid name) produces
- 3287 unspecified results. The form *symbol=value* is used by the KornShell *name[expression]=value*
- 3288 syntax.

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

(blank page)

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

File Format Notation

The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility descriptions use a syntax to describe the data organization within the files, when that organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces volume of POSIX.1-2008 *printf()* function, as described in this chapter. When used in STDIN or INPUT FILES sections of the utility descriptions, this syntax describes the format that could have been used to write the text to be read, not a format that could be used by the System Interfaces volume of POSIX.1-2008 *scanf()* function to read the input file.

The description of an individual record is as follows:

"<format>", [<arg1>, <arg2>, ..., <argn>]

The *format* is a character string that contains three types of objects defined below:

1. *Characters* that are not "escape sequences" or "conversion specifications", as described below, shall be copied to the output.
2. *Escape Sequences* represent non-graphic characters.
3. *Conversion Specifications* specify the output format of each argument; see below.

The following characters have the following special meaning in the format string:

' ' (An empty character position.) Represents one or more <blank> characters.

Δ Represents exactly one <space> character.

Table 5-1 lists escape sequences and associated actions on display devices capable of the action.

Table 5-1 Escape Sequences and Associated Actions

Escape Sequence	Represents Character	Terminal Action
\\	<backslash>	Print the <backslash> character.
\a	<alert>	Attempt to alert the user through audible or visible notification.
\b	<backspace>	Move the printing position to one column before the current position, unless the current position is the start of a line.
\f	<form-feed>	Move the printing position to the initial printing position of the next logical page.
\n	<newline>	Move the printing position to the start of the next line.
\r	<carriage-return>	Move the printing position to the start of the current line.
\t	<tab>	Move the printing position to the next tab position on the current line. If there are no more tab positions remaining on the line, the behavior is undefined.
\v	<vertical-tab>	Move the printing position to the start of the next <vertical-tab> position. If there are no more <vertical-tab> positions left on the page, the behavior is undefined.

3326		Each conversion specification is introduced by the <percent-sign> character ('%'). After the
3327		character '%', the following shall appear in sequence:
3328	<i>flags</i>	Zero or more <i>flags</i> , in any order, that modify the meaning of the conversion
3329		specification.
3330	<i>field width</i>	An optional string of decimal digits to specify a minimum field width. For an
3331		output field, if the converted value has fewer bytes than the field width, it shall be
3332		padded on the left (or right, if the left-adjustment flag ('-'), described below, has
3333		been given) to the field width.
3334	<i>precision</i>	Gives the minimum number of digits to appear for the d, o, i, u, x, or X conversion
3335		specifiers (the field is padded with leading zeros), the number of digits to appear
3336		after the radix character for the e and f conversion specifiers, the maximum
3337		number of significant digits for the g conversion specifier; or the maximum
3338		number of bytes to be written from a string in the s conversion specifier. The
3339		precision shall take the form of a <period> ('.') followed by a decimal digit
3340		string; a null digit string is treated as zero.
3341	<i>conversion specifier characters</i>	
3342		A conversion specifier character (see below) that indicates the type of conversion
3343		to be applied.
3344		The <i>flag</i> characters and their meanings are:
3345	-	The result of the conversion shall be left justified within the field.
3346	+	The result of a signed conversion shall always begin with a sign ('+' or '-').
3347	<space>	If the first character of a signed conversion is not a sign, a <space> shall be
3348		prefixed to the result. This means that if the <space> and '+' flags both appear,
3349		the <space> flag shall be ignored.
3350	#	The value shall be converted to an alternative form. For c, d, i, u, and s
3351		conversion specifiers, the behavior is undefined. For the o conversion specifier, it
3352		shall increase the precision to force the first digit of the result to be a zero. For x or
3353		X conversion specifiers, a non-zero result has 0x or 0X prefixed to it, respectively.
3354		For a, A, e, E, f, F, g, and G conversion specifiers, the result shall always contain a
3355		radix character, even if no digits follow the radix character. For g and G conversion
3356		specifiers, trailing zeros shall not be removed from the result as they usually are.
3357	0	For a, A, d, e, E, f, F, g, G, i, o, u, x, and X conversion specifiers, leading zeros
3358		(following any indication of sign or base) shall be used to pad to the field width
3359		rather than performing space padding, except when converting an infinity or NaN.
3360		If the '0' and '-' flags both appear, the '0' flag shall be ignored. For d, i, o, u,
3361		x, and X conversion specifiers, if a precision is specified, the '0' flag shall be
3362		ignored. For other conversion specifiers, the behavior is undefined.
3363		Each conversion specifier character shall result in fetching zero or more arguments. The results
3364		are undefined if there are insufficient arguments for the format. If the format is exhausted while
3365		arguments remain, the excess arguments shall be ignored.
3366		The conversion specifiers and their meanings are:
3367	a, A	The floating-point number argument representing a floating-point number shall be
3368		converted in the style "[-] 0xh.hhhhp±d", where there is one hexadecimal digit
3369		(which shall be non-zero if the argument is a normalized floating-point number
3370		and is otherwise unspecified) before the decimal-point character and the number
3371		of hexadecimal digits after it is equal to the precision; if the precision is missing

File Format Notation

3372		and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact
3373		representation of the value; if the precision is missing and FLT_RADIX is not a
3374		power of 2, then the precision shall be sufficient to distinguish different floating-
3375		point values in the internal representation used by the utility, except that trailing
3376		zeros may be omitted; if the precision is zero and the # flag is not specified, no
3377		decimal-point character shall appear. The letters "abcdef" shall be used for a
3378		conversion and the letters "ABCDEF" for A conversion. The A conversion specifier
3379		produces a number with X and P instead of x and p. The exponent shall always
3380		contain at least one digit, and only as many more digits as necessary to represent
3381		the decimal exponent of 2. If the value is zero, the exponent shall be zero. A
3382		floating-point number argument representing an infinity or NaN shall be
3383		converted in the style of an f or F conversion specifier.
3384	d,i,o,u,x,X	The integer argument shall be written as signed decimal (d or i), unsigned octal
3385		(o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and
3386		i specifiers shall convert to signed decimal in the style "[−]dddd". The x
3387		conversion specifier shall use the numbers and letters "0123456789abcdef" and
3388		the X conversion specifier shall use the numbers and letters
3389		"0123456789ABCDEF". The <i>precision</i> component of the argument shall specify
3390		the minimum number of digits to appear. If the value being converted can be
3391		represented in fewer digits than the specified minimum, it shall be expanded with
3392		leading zeros. The default precision shall be 1. The result of converting a zero
3393		value with a precision of 0 shall be no characters. If both the field width and
3394		precision are omitted, the implementation may precede, follow, or precede and
3395		follow numeric arguments of types d, i, and u with <blank> characters; arguments
3396		of type o (octal) may be preceded with leading zeros.
3397	f,F	The floating-point number argument shall be written in decimal notation in the
3398		style "[−]ddd.ddd", where the number of digits after the radix character (shown here
3399		as a decimal point) shall be equal to the <i>precision</i> specification. The LC_NUMERIC
3400		locale category shall determine the radix character to use in this format. If the
3401		<i>precision</i> is omitted from the argument, six digits shall be written after the radix
3402		character; if the <i>precision</i> is explicitly 0, no radix character shall appear.
3403		A floating-point number argument representing an infinity shall be converted in
3404		one of the styles "[−]inf" or "[−]infinity"; which style is implementation-
3405		defined. A floating-point number argument representing a NaN shall be converted
3406		in one of the styles "[−]nan(<i>n-char-sequence</i>)" or "[−]nan"; which style,
3407		and the meaning of any <i>n-char-sequence</i> , is implementation-defined. The F
3408		conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf",
3409		"infinity", or "nan", respectively.
3410	e,E	The floating-point number argument shall be written in the style "[−]d.ddde±dd (the
3411		symbol '±' indicates either a <plus-sign> or minus-sign), where there is one digit
3412		before the radix character (shown here as a decimal point) and the number of
3413		digits after it is equal to the precision. The LC_NUMERIC locale category shall
3414		determine the radix character to use in this format. When the precision is missing,
3415		six digits shall be written after the radix character; if the precision is 0, no radix
3416		character shall appear. The E conversion specifier shall produce a number with E
3417		instead of e introducing the exponent. The exponent shall always contain at least
3418		two digits. However, if the value to be written requires an exponent greater than
3419		two digits, additional exponent digits shall be written as necessary.
3420		A floating-point number argument representing an infinity or NaN shall be
3421		converted in the style of an f or F conversion specifier.

3422	g, G	The floating-point number argument shall be written in style f or e (or in style F or E in the case of a G conversion specifier), with the precision specifying the number of significant digits. The style used depends on the value converted: style e (or E) shall be used only if the exponent resulting from the conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character shall appear only if it is followed by a digit.
3423		
3424		
3425		
3426		
3427		
3428		A floating-point number argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.
3429		
3430	c	The single-byte character argument shall be written.
3431	s	The argument shall be taken to be a string and bytes from the string shall be written until the end of the string or the number of bytes indicated by the <i>precision</i> specification of the argument is reached. If the precision is omitted from the argument, it shall be taken to be infinite, so all bytes up to the end of the string shall be written.
3432		
3433		
3434		
3435		
3436	$\%$	Write a ' $\%$ ' character; no argument is converted.
3437		In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of
3438		a conversion is wider than the field width, the field is simply expanded to contain the
3439		conversion result. The term "field width" should not be confused with the term "precision"
3440		used in the description of $\%s$.

Examples

To represent the output of a program that prints a date and time in the form Sunday, July 3, 10:02, where *weekday* and *month* are strings:

```
"%s, %s %d, %d: %.2d\n" <weekday>, <month>, <day>, <hour>, <min>
```

To show ' π ' written to 5 decimal places:

```
"pi = %.5f\n", <value of  $\pi$ >
```

To show an input file format consisting of five <colon>-separated fields:

```
"%s: %s: %s: %s: %s\n", <arg1>, <arg2>, <arg3>, <arg4>, <arg5>
```


6.1 Portable Character Set

Conforming implementations shall support one or more coded character sets. Each supported locale shall include the *portable character set*, which is the set of symbolic names for characters in Table 6-1. This is used to describe characters within the text of POSIX.1-2008. The first eight entries in Table 6-1 are defined in the ISO/IEC 6429:1992 standard and the rest of the characters are defined in the ISO/IEC 10646-1:2000 standard.

Table 6-1 Portable Character Set

Symbolic Name	Glyph	UCS	Description
<NUL>		<U0000>	NULL (NUL)
<alert>		<U0007>	BELL (BEL)
<backspace>		<U0008>	BACKSPACE (BS)
<tab>		<U0009>	CHARACTER TABULATION (HT)
<carriage-return>		<U000D>	CARRIAGE RETURN (CR)
<newline>		<U000A>	LINE FEED (LF)
<vertical-tab>		<U000B>	LINE TABULATION (VT)
<form-feed>		<U000C>	FORM FEED (FF)
<space>		<U0020>	SPACE
<exclamation-mark>	!	<U0021>	EXCLAMATION MARK
<quotation-mark>	"	<U0022>	QUOTATION MARK
<number-sign>	#	<U0023>	NUMBER SIGN
<dollar-sign>	\$	<U0024>	DOLLAR SIGN
<percent-sign>	%	<U0025>	PERCENT SIGN
<ampersand>	&	<U0026>	AMPERSAND
<apostrophe>	'	<U0027>	APOSTROPHE
<left-parenthesis>	(	<U0028>	LEFT PARENTHESIS
<right-parenthesis>	)	<U0029>	RIGHT PARENTHESIS
<asterisk>	*	<U002A>	ASTERISK
<plus-sign>	+	<U002B>	PLUS SIGN
<comma>	,	<U002C>	COMMA
<hyphen-minus>	–	<U002D>	HYPHEN-MINUS
<hyphen>	–	<U002D>	HYPHEN-MINUS
<full-stop>	.	<U002E>	FULL STOP
<period>	.	<U002E>	FULL STOP
<slash>	/	<U002F>	SOLIDUS
<solidus>	/	<U002F>	SOLIDUS
<zero>	0	<U0030>	DIGIT ZERO
<one>	1	<U0031>	DIGIT ONE
<two>	2	<U0032>	DIGIT TWO

	Symbolic Name	Glyph	UCS	Description
3489				
3490	<three>	3	<U0033>	DIGIT THREE
3491	<four>	4	<U0034>	DIGIT FOUR
3492	<five>	5	<U0035>	DIGIT FIVE
3493	<six>	6	<U0036>	DIGIT SIX
3494	<seven>	7	<U0037>	DIGIT SEVEN
3495	<eight>	8	<U0038>	DIGIT EIGHT
3496	<nine>	9	<U0039>	DIGIT NINE
3497	<colon>	:	<U003A>	COLON
3498	<semicolon>	;	<U003B>	SEMICOLON
3499	<less-than-sign>	<	<U003C>	LESS-THAN SIGN
3500	<equals-sign>	=	<U003D>	EQUALS SIGN
3501	<greater-than-sign>	>	<U003E>	GREATER-THAN SIGN
3502	<question-mark>	?	<U003F>	QUESTION MARK
3503	<commercial-at>	@	<U0040>	COMMERCIAL AT
3504	<A>	A	<U0041>	LATIN CAPITAL LETTER A
3505		B	<U0042>	LATIN CAPITAL LETTER B
3506	<C>	C	<U0043>	LATIN CAPITAL LETTER C
3507	<D>	D	<U0044>	LATIN CAPITAL LETTER D
3508	<E>	E	<U0045>	LATIN CAPITAL LETTER E
3509	<F>	F	<U0046>	LATIN CAPITAL LETTER F
3510	<G>	G	<U0047>	LATIN CAPITAL LETTER G
3511	<H>	H	<U0048>	LATIN CAPITAL LETTER H
3512	<I>	I	<U0049>	LATIN CAPITAL LETTER I
3513	<J>	J	<U004A>	LATIN CAPITAL LETTER J
3514	<K>	K	<U004B>	LATIN CAPITAL LETTER K
3515	<L>	L	<U004C>	LATIN CAPITAL LETTER L
3516	<M>	M	<U004D>	LATIN CAPITAL LETTER M
3517	<N>	N	<U004E>	LATIN CAPITAL LETTER N
3518	<O>	O	<U004F>	LATIN CAPITAL LETTER O
3519	<P>	P	<U0050>	LATIN CAPITAL LETTER P
3520	<Q>	Q	<U0051>	LATIN CAPITAL LETTER Q
3521	<R>	R	<U0052>	LATIN CAPITAL LETTER R
3522	<S>	S	<U0053>	LATIN CAPITAL LETTER S
3523	<T>	T	<U0054>	LATIN CAPITAL LETTER T
3524	<U>	U	<U0055>	LATIN CAPITAL LETTER U
3525	<V>	V	<U0056>	LATIN CAPITAL LETTER V
3526	<W>	W	<U0057>	LATIN CAPITAL LETTER W
3527	<X>	X	<U0058>	LATIN CAPITAL LETTER X
3528	<Y>	Y	<U0059>	LATIN CAPITAL LETTER Y
3529	<Z>	Z	<U005A>	LATIN CAPITAL LETTER Z
3530	<left-square-bracket>	[	<U005B>	LEFT SQUARE BRACKET
3531	<backslash>	\	<U005C>	REVERSE SOLIDUS
3532	<reverse-solidus>	\	<U005C>	REVERSE SOLIDUS
3533	<right-square-bracket>	]	<U005D>	RIGHT SQUARE BRACKET
3534	<circumflex-accent>	^	<U005E>	CIRCUMFLEX ACCENT
3535	<circumflex>	^	<U005E>	CIRCUMFLEX ACCENT
3536	<low-line>	—	<U005F>	LOW LINE
3537	<underscore>	—	<U005F>	LOW LINE
3538	<grave-accent>	`	<U0060>	GRAVE ACCENT
3539	<a>	a	<U0061>	LATIN SMALL LETTER A
3540		b	<U0062>	LATIN SMALL LETTER B

Character Set

Portable Character Set

Symbolic Name	Glyph	UCS	Description
<c>	c	<U0063>	LATIN SMALL LETTER C
<d>	d	<U0064>	LATIN SMALL LETTER D
<e>	e	<U0065>	LATIN SMALL LETTER E
<f>	f	<U0066>	LATIN SMALL LETTER F
<g>	g	<U0067>	LATIN SMALL LETTER G
<h>	h	<U0068>	LATIN SMALL LETTER H
<i>	i	<U0069>	LATIN SMALL LETTER I
<j>	j	<U006A>	LATIN SMALL LETTER J
<k>	k	<U006B>	LATIN SMALL LETTER K
<l>	l	<U006C>	LATIN SMALL LETTER L
<m>	m	<U006D>	LATIN SMALL LETTER M
<n>	n	<U006E>	LATIN SMALL LETTER N
<o>	o	<U006F>	LATIN SMALL LETTER O
<p>	p	<U0070>	LATIN SMALL LETTER P
<q>	q	<U0071>	LATIN SMALL LETTER Q
<r>	r	<U0072>	LATIN SMALL LETTER R
<s>	s	<U0073>	LATIN SMALL LETTER S
<t>	t	<U0074>	LATIN SMALL LETTER T
<u>	u	<U0075>	LATIN SMALL LETTER U
<v>	v	<U0076>	LATIN SMALL LETTER V
<w>	w	<U0077>	LATIN SMALL LETTER W
<x>	x	<U0078>	LATIN SMALL LETTER X
<y>	y	<U0079>	LATIN SMALL LETTER Y
<z>	z	<U007A>	LATIN SMALL LETTER Z
<left-brace>	{	<U007B>	LEFT CURLY BRACKET
<left-curly-bracket>	{	<U007B>	LEFT CURLY BRACKET
<vertical-line>		<U007C>	VERTICAL LINE
<right-brace>	}	<U007D>	RIGHT CURLY BRACKET
<right-curly-bracket>	}	<U007D>	RIGHT CURLY BRACKET
<tilde>	~	<U007E>	TILDE

POSIX.1-2008 uses character names other than the above, but only in an informative way; for example, in examples to illustrate the use of characters beyond the portable character set with the facilities of POSIX.1-2008.

Table 6-1 (on page 125) defines the characters in the portable character set and the corresponding symbolic character names used to identify each character in a character set description file. The table contains more than one symbolic character name for characters whose traditional name differs from the chosen name. Characters defined in **Table 6-2** (on page 130) may also be used in character set description files.

POSIX.1-2008 places only the following requirements on the encoded values of the characters in the portable character set:

- If the encoded values associated with each member of the portable character set are not invariant across all locales supported by the implementation, if an application accesses any pair of locales where the character encodings differ, or accesses data from an application running in a locale which has different encodings from the application's current locale, the results are unspecified.
- The encoded values associated with the digits 0 to 9 shall be such that the value of each character after 0 shall be one greater than the value of the previous character.

- 3589 • A null character, NUL, which has all bits set to zero, shall be in the set of characters.
 - 3590 • The encoded values associated with the members of the portable character set are each
 - 3591 represented in a single byte. Moreover, if the value is stored in an object of C-language
 - 3592 type **char**, it is guaranteed to be positive (except the NUL, which is always zero).
- 3593 Conforming implementations shall support certain character and character set attributes, as
- 3594 defined in [Section 7.2](#) (on page 136).

3595 6.2 Character Encoding

3596 The POSIX locale contains the characters in [Table 6-1](#) (on page 125), which have the properties

3597 listed in [Section 7.3.1](#) (on page 139). In other locales, the presence, meaning, and representation

3598 of any additional characters are locale-specific.

3599 In locales other than the POSIX locale, a character may have a state-dependent encoding. There

3600 are two types of these encodings:

- 3601 • A single-shift encoding (where each character not in the initial shift state is preceded by a
- 3602 shift code) can be defined if each shift-code and character sequence is considered a multi-
- 3603 byte character. This is done using the concatenated-constant format in a character set
- 3604 description file, as described in [Section 6.4](#) (on page 129). If the implementation supports a
- 3605 character encoding of this type, all of the standard utilities in the Shell and Utilities volume
- 3606 of POSIX.1-2008 shall support it. Use of a single shift encoding with any of the functions in
- 3607 the System Interfaces volume of POSIX.1-2008 that do not specifically mention the effects
- 3608 of state-dependent encoding is implementation-defined.
- 3609 • A locking-shift encoding (where the state of the character is determined by a shift code
- 3610 that may affect more than the single character following it) cannot be defined with the
- 3611 current character set description file format. Use of a locking-shift encoding with any of
- 3612 the standard utilities in the Shell and Utilities volume of POSIX.1-2008 or with any of the
- 3613 functions in the System Interfaces volume of POSIX.1-2008 that do not specifically mention
- 3614 the effects of state-dependent encoding is implementation-defined.

3615 While in the initial shift state, all characters in the portable character set shall retain their usual

3616 interpretation and shall not alter the shift state. The interpretation for subsequent bytes in the

3617 sequence shall be a function of the current shift state. A byte with all bits zero shall be

3618 interpreted as the null character independent of shift state. Such a byte shall not occur as part of

3619 any other character.

3620 The maximum allowable number of bytes in a character in the current locale shall be indicated

3621 by {MB_CUR_MAX}, defined in the [<stdlib.h>](#) header and by the [<mb_cur_max>](#) value in a

3622 character set description file; see [Section 6.4](#) (on page 129). The implementation's maximum

3623 number of bytes in a character shall be defined by the C-language macro {MB_LEN_MAX}.

6.3 C Language Wide-Character Codes

In the shell, the standard utilities are written so that the encodings of characters are described by the locale's `LC_CTYPE` definition (see [Section 7.3.1](#), on page 139) and there is no differentiation between characters consisting of single octets (8-bit bytes) or multiple bytes. However, in the C language, a differentiation is made. To ease the handling of variable length characters, the C language has introduced the concept of wide-character codes.

All wide-character codes in a given process consist of an equal number of bits. This is in contrast to characters, which can consist of a variable number of bytes. The byte or byte sequence that represents a character can also be represented as a wide-character code. Wide-character codes thus provide a uniform size for manipulating text data. A wide-character code having all bits zero is the null wide-character code (see [Section 3.246](#), on page 72), and terminates wide-character strings (see [Section 3.435](#), on page 103). The wide-character value for each member of the portable character set shall equal its value when used as the lone character in an integer character constant. Wide-character codes for other characters are locale and implementation-defined. State shift bytes shall not have a wide-character code representation. POSIX.1-2008 provides no means of defining a wide-character codeset.

6.4 Character Set Description File

Implementations shall provide a character set description file for at least one coded character set supported by the implementation. These files are referred to elsewhere in POSIX.1-2008 as *charmap* files. It is implementation-defined whether or not users or applications can provide additional character set description files.

POSIX.1-2008 does not require that multiple character sets or codesets be supported. Although multiple *charmap* files are supported, it is the responsibility of the implementation to provide the file or files; if only one is provided, only that one is accessible using the *localedef* utility's *-f* option.

Each character set description file, except those that use the ISO/IEC 10646-1:2000 standard position values as the encoding values, shall define characteristics for the coded character set and the encoding for the characters specified in [Table 6-1](#) (on page 125), and may define encoding for additional characters supported by the implementation. Other information about the coded character set may also be in the file. Coded character set character values shall be defined using symbolic character names followed by character encoding values.

Each symbolic name specified in [Table 6-1](#) (on page 125) shall be included in the file and shall be mapped to a unique coding value, except as noted below. The glyphs represented by the C character constants `'{'`, `'}'`, `'_'`, `'-'`, `'/'`, `'\\'`, `'.'`, and `'^'` have more than one symbolic name; all symbolic names for each such glyph shall be included, each with identical encoding. If some or all of the control characters identified in [Table 6-2](#) (on page 130) are supported by the implementation, the symbolic names and their corresponding encoding values shall be included in the file. Some of the encodings associated with the symbolic names in [Table 6-2](#) (on page 130) may be the same as characters found in [Table 6-1](#) (on page 125); both names shall be provided for each encoding.

Table 6-2 Control Character Set

3664

3665	<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>
3666	<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>
3667	<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>
3668	<CAN>	<DEL>	<ETB>	<IS1>	<RS>	<SYN>
3669	<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>
3670	<DC1>		<FF>	<IS3>	<SO>	<VT>

3671 The following declarations can precede the character definitions. Each shall consist of the
 3672 symbol shown in the following list, starting in column 1, including the surrounding brackets,
 3673 followed by one or more <blank> characters, followed by the value to be assigned to the symbol.

3674 **<code_set_name>** The name of the coded character set for which the character set
 3675 description file is defined. The characters of the name shall be taken from
 3676 the set of characters with visible glyphs defined in Table 6-1 (on page
 3677 125).

3678 **<mb_cur_max>** The maximum number of bytes in a multi-byte character. This shall
 3679 default to 1.

3680 **<mb_cur_min>** An unsigned positive integer value that defines the minimum number of
 3681 XSI bytes in a character for the encoded character set. On XSI-conformant
 3682 systems, <mb_cur_min> shall always be 1.

3683 **<escape_char>** The character used to indicate that the characters following shall be
 3684 interpreted in a special way, as defined later in this section. This shall
 3685 default to <backslash> ('\\'), which is the character used in all the
 3686 following text and examples, unless otherwise noted.

3687 **<comment_char>** The character that, when placed in column 1 of a charmap line, is used to
 3688 indicate that the line shall be ignored. The default character shall be the
 3689 <number-sign> ('#').

3690 The character set mapping definitions shall be all the lines immediately following an identifier
 3691 line containing the string "CHARMAP" starting in column 1, and preceding a trailer line
 3692 containing the string "END CHARMAP" starting in column 1. Empty lines and lines containing a
 3693 <comment_char> in the first column shall be ignored. Each non-comment line of the character
 3694 set mapping definition (that is, between the "CHARMAP" and "END CHARMAP" lines of the file)
 3695 shall be in either of two forms:

3696 "%s %s %s\n", <symbolic-name>, <encoding>, <comments>

3697 or:

3698 "%s...%s %s %s\n", <symbolic-name>, <symbolic-name>,
 3699 <encoding>, <comments>

3700 In the first format, the line in the character set mapping definition shall define a single symbolic
 3701 name and a corresponding encoding. A symbolic name is one or more characters from the set
 3702 shown with visible glyphs in Table 6-1 (on page 125), enclosed between angle brackets. A
 3703 character following an escape character is interpreted as itself; for example, the sequence
 3704 "<\\>>" represents the symbolic name ">" enclosed between angle brackets.

3705 In the second format, the line in the character set mapping definition shall define a range of one
 3706 or more symbolic names. In this form, the symbolic names shall consist of zero or more non-
 3707 numeric characters from the set shown with visible glyphs in Table 6-1 (on page 125), followed
 3708 by an integer formed by one or more decimal digits. Both integers shall contain the same

Character Set

Character Set Description File

number of digits. The characters preceding the integer shall be identical in the two symbolic names, and the integer formed by the digits in the second symbolic name shall be equal to or greater than the integer formed by the digits in the first name. This shall be interpreted as a series of symbolic names formed from the common part and each of the integers between the first and the second integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the symbolic names <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

A character set mapping definition line shall exist for all symbolic names specified in Table 6-1 (on page 125), and shall define the coded character value that corresponds to the character indicated in the table, or the coded character value that corresponds to the control character symbolic name. If the control characters commonly associated with the symbolic names in Table 6-2 (on page 130) are supported by the implementation, the symbolic name and the corresponding encoding value shall be included in the file. Additional unique symbolic names may be included. A coded character value can be represented by more than one symbolic name.

The encoding part is expressed as one (for single-byte character values) or more concatenated decimal, octal, or hexadecimal constants in the following formats:

```
"%cd%u", <escape_char>, <decimal byte value>
```

```
"%cx%x", <escape_char>, <hexadecimal byte value>
```

```
"%co", <escape_char>, <octal byte value>
```

Decimal constants shall be represented by two or three decimal digits, preceded by the escape character and the lowercase letter 'd'; for example, "\d05", "\d97", or "\d143". Hexadecimal constants shall be represented by two hexadecimal digits, preceded by the escape character and the lowercase letter 'x'; for example, "\x05", "\x61", or "\x8f". Octal constants shall be represented by two or three octal digits, preceded by the escape character; for example, "\05", "\141", or "\217". In a portable charmap file, each constant represents an 8-bit byte. When constants are concatenated for multi-byte character values, they shall be of the same type, and interpreted in sequence from first to last with the first byte of the multi-byte character specified by the first byte in the sequence. The manner in which these constants are represented in the character stored in the system is implementation-defined. (This notation was chosen for reasons of portability. There is no requirement that the internal representation in the computer memory be in this same order.) Omitting bytes from a multi-byte character definition produces undefined results.

In lines defining ranges of symbolic names, the encoded value shall be the value for the first symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic names defined by the range shall have encoding values in increasing order. Bytes shall be treated as unsigned octets, and carry shall be propagated between the bytes as necessary to represent the range. However, because this causes a null byte in the second or subsequent bytes of a character, such a declaration should not be specified. For example, the line:

```
<j0101>...<j0104> \d129\d254
```

is interpreted as:

```
<j0101> \d129\d254
```

```
<j0102> \d129\d255
```

```
<j0103> \d130\d00
```

```
<j0104> \d130\d01
```

The expanded declaration of the symbol <j0103> in the above example is an invalid specification, because it contains a null byte in the second byte of a character.

The comment is optional.

POSIX.1-2008 provides no means of defining a wide-character codeset.

The following declarations can follow the character set mapping definitions (after the "END CHARMAP" statement). Each shall consist of the keyword shown in the following list, starting in column 1, followed by the value(s) to be associated to the keyword, as defined below.

WIDTH A non-negative integer value defining the column width (see [Section 3.103](#), on page 50) for the printable characters in the coded character set specified in [Table 6-1](#) (on page 125) and [Table 6-2](#) (on page 130). Coded character set character values shall be defined using symbolic character names followed by column width values. Defining a character with more than one **WIDTH** produces undefined results. The **END WIDTH** keyword shall be used to terminate the **WIDTH** definitions. Specifying the width of a non-printable character in a **WIDTH** declaration produces undefined results.

WIDTH_DEFAULT

A non-negative integer value defining the default column width for any printable character not listed by one of the **WIDTH** keywords. If no **WIDTH_DEFAULT** keyword is included in the charmap, the default character width shall be 1.

Example

After the "END CHARMAP" statement, a syntax for a width definition would be:

```
WIDTH
<A> 1
<B> 1
<C>...<Z> 1
...
<fool>...<foom> 2
...
END WIDTH
```

In this example, the numerical code point values represented by the symbols **<A>** and **** are assigned a width of 1. The code point values **<C>** to **<Z>** inclusive (**<C>**, **<D>**, **<E>**, and so on) are also assigned a width of 1. Using **<A>...<Z>** would have required fewer lines, but the alternative was shown to demonstrate flexibility. The keyword **WIDTH_DEFAULT** could have been added as appropriate.

6.4.1 State-Dependent Character Encodings

This section addresses the use of state-dependent character encodings (that is, those in which the encoding of a character is dependent on one or more shift codes that may precede it).

A single-shift encoding (where each character not in the initial shift state is preceded by a shift code) can be defined in the charmap format if each shift-code/character sequence is considered a multi-byte character, defined using the concatenated-constant format described in [Section 6.4](#) (on page 129). If the implementation supports a character encoding of this type, all of the standard utilities shall support it. A locking-shift encoding (where the state of the character is determined by a shift code that may affect more than the single character following it) could be defined with an extension to the charmap format described in [Section 6.4](#) (on page 129).

*Character Set**Character Set Description File*

3796 If the implementation supports a character encoding of this type, any of the standard utilities
3797 that describe character (*versus* byte) or text-file manipulation shall have the following
3798 characteristics:

- 3799 1. The utility shall process the statefully encoded data as a concatenation of state-
3800 independent characters. The presence of redundant locking shifts shall not affect the
3801 comparison of two statefully encoded strings.
- 3802 2. A utility that divides, truncates, or extracts substrings from statefully encoded data shall
3803 produce output that contains locking shifts at the beginning or end of the resulting data,
3804 if appropriate, to retain correct state information.

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

(blank page)

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

7.1 General

A locale is the definition of the subset of a user's environment that depends on language and cultural conventions. It is made up from one or more categories. Each category is identified by its name and controls specific aspects of the behavior of components of the system. Category names correspond to the following environment variable names:

LC_CTYPE Character classification and case conversion.

LC_COLLATE Collation order.

LC_MONETARY Monetary formatting.

LC_NUMERIC Numeric, non-monetary formatting.

LC_TIME Date and time formats.

LC_MESSAGES Formats of informative and diagnostic messages and interactive responses.

The standard utilities in the Shell and Utilities volume of POSIX.1-2008 shall base their behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each utility. The behavior of some of the C-language functions defined in the System Interfaces volume of POSIX.1-2008 shall also be modified based on the current locale, as defined by the last call to *setlocale()*.

Locales other than those supplied by the implementation can be created via the *localedef* utility, provided that the *_POSIX2_LOCALEDEF* symbol is defined on the system. Even if *localedef* is not provided, all implementations conforming to the System Interfaces volume of POSIX.1-2008 shall provide one or more locales that behave as described in this chapter. The input to the utility is described in Section 7.3 (on page 136). The value that is used to specify a locale when using environment variables shall be the string specified as the *name* operand to the *localedef* utility when the locale was created. The strings "C" and "POSIX" are reserved as identifiers for the POSIX locale (see Section 7.2, on page 136). When the value of a locale environment variable begins with a <slash> (' / '), it shall be interpreted as the pathname of the locale definition; the type of file (regular, directory, and so on) used to store the locale definition is implementation-defined. If the value does not begin with a <slash>, the mechanism used to locate the locale is implementation-defined.

If different character sets are used by the locale categories, the results achieved by an application utilizing these categories are undefined. Likewise, if different codesets are used for the data being processed by interfaces whose behavior is dependent on the current locale, or the codeset is different from the codeset assumed when the locale was created, the result is also undefined.

Applications can select the desired locale by invoking the *setlocale()* function (or equivalent) with the appropriate value. If the function is invoked with an empty string, such as:

```
setlocale(LC_ALL, "");
```

the value of the corresponding environment variable is used. If the environment variable is

unset or is set to the empty string, the implementation shall set the appropriate environment as defined in [Chapter 8](#) (on page 173).

7.2 POSIX Locale

Conforming systems shall provide a POSIX locale, also known as the C locale. The behavior of standard utilities and functions in the POSIX locale shall be as if the locale was defined via the *localedef* utility with input data from the POSIX locale tables in [Section 7.3](#).

The tables in [Section 7.3](#) describe the characteristics and behavior of the POSIX locale for data consisting entirely of characters from the portable character set and the control character set. For other characters, the behavior is unspecified. For C-language programs, the POSIX locale shall be the default locale when the *setlocale()* function is not called.

The POSIX locale can be specified by assigning to the appropriate environment variables the values "C" or "POSIX".

All implementations shall define a locale as the default locale, to be invoked when no environment variables are set, or set to the empty string. This default locale can be the POSIX locale or any other implementation-defined locale. Some implementations may provide facilities for local installation administrators to set the default locale, customizing it for each location. POSIX.1-2008 does not require such a facility.

7.3 Locale Definition

The capability to specify additional locales to those provided by an implementation is optional, denoted by the `_POSIX2_LOCALEDEF` symbol. If the option is not supported, only implementation-supplied locales are available. Such locales shall be documented using the format specified in this section.

Locales can be described with the file format presented in this section. The file format is that accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the "locale definition file", but no locales shall be affected by this file unless it is processed by *localedef* or some similar mechanism. Any requirements in this section imposed upon the utility shall apply to *localedef* or to any other similar utility used to install locale information using the locale definition file format described here.

The locale definition file shall contain one or more locale category source definitions, and shall not contain more than one definition for the same locale category. If the file contains source definitions for more than one category, implementation-defined categories, if present, shall appear after the categories defined by [Section 7.1](#) (on page 135). A category source definition contains either the definition of a category or a **copy** directive. For a description of the **copy** directive, see *localedef*. In the event that some of the information for a locale category, as specified in this volume of POSIX.1-2008, is missing from the locale source definition, the behavior of that category, if it is referenced, is unspecified.

A category source definition shall consist of a category header, a category body, and a category trailer. A category header shall consist of the character string naming of the category, beginning with the characters `LC_`. The category trailer shall consist of the string "END", followed by one or more <blank> characters and the string used in the corresponding category header.

The category body shall consist of one or more lines of text. Each line shall contain an identifier, optionally followed by one or more operands. Identifiers shall be either keywords, identifying a

Locale

Locale Definition

particular locale element, or collating elements. In addition to the keywords defined in this volume of POSIX.1-2008, the source can contain implementation-defined keywords. Each keyword within a locale shall have a unique name (that is, two categories cannot have a commonly-named keyword); no keyword shall start with the characters *LC_*. Identifiers shall be separated from the operands by one or more <blank> characters.

Operands shall be characters, collating elements, or strings of characters. Strings shall be enclosed in double-quotes. Literal double-quotes within strings shall be preceded by the <escape character>, described below. When a keyword is followed by more than one operand, the operands shall be separated by <semicolon> characters; <blank> characters shall be allowed both before and after a <semicolon>.

The first category header in the file can be preceded by a line modifying the comment character. It shall have the following format, starting in column 1:

```
"comment_char %c\n", <comment character>
```

The comment character shall default to the <number-sign> ('#'). Blank lines and lines containing the <comment character> in the first position shall be ignored.

The first category header in the file can be preceded by a line modifying the escape character to be used in the file. It shall have the following format, starting in column 1:

```
"escape_char %c\n", <escape character>
```

The escape character shall default to <backslash>, which is the character used in all examples shown in this volume of POSIX.1-2008.

A line can be continued by placing an escape character as the last character on the line; this continuation character shall be discarded from the input. Although the implementation need not accept any one portion of a continued line with a length exceeding {LINE_MAX} bytes, it shall place no limits on the accumulated length of the continued line. Comment lines shall not be continued on a subsequent line using an escaped <newline>.

Individual characters, characters in strings, and collating elements shall be represented using symbolic names, as defined below. In addition, characters can be represented using the characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic notation is used, the resultant locale definitions are in many cases not portable between systems. The left angle bracket ('<') is a reserved symbol, denoting the start of a symbolic name; when used to represent itself it shall be preceded by the escape character. The following rules apply to character representation:

1. A character can be represented via a symbolic name, enclosed within angle brackets '<' and '>'. The symbolic name, including the angle brackets, shall exactly match a symbolic name defined in the charmap file specified via the *localedef* -f option, and it shall be replaced by a character value determined from the value associated with the symbolic name in the charmap file. The use of a symbolic name not found in the charmap file shall constitute an error, unless the category is *LC_CTYPE* or *LC_COLLATE*, in which case it shall constitute a warning condition (see *localedef* for a description of actions resulting from errors and warnings). The specification of a symbolic name in a **collating-element** or **collating-symbol** section that duplicates a symbolic name in the charmap file (if present) shall be an error. Use of the escape character or a right angle bracket within a symbolic name is invalid unless the character is preceded by the escape character.

For example:

```
<c>;<c-cedilla> "<M><a><y>"
```

Locale Definition

Locale

2. A character in the portable character set can be represented by the character itself, in which case the value of the character is implementation-defined. (Implementations may allow other characters to be represented as themselves, but such locale definitions are not portable.) Within a string, the double-quote character, the escape character, and the right angle bracket character shall be escaped (preceded by the escape character) to be interpreted as the character itself. Outside strings, the characters:

, ; < > *escape_char*

shall be escaped to be interpreted as the character itself.

For example:

c "May"

3. A character can be represented as an octal constant. An octal constant shall be specified as the escape character followed by two or three octal digits. Each constant shall represent a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

For example:

\143;\347;\143\150 "\115\141\171"

4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall be specified as the escape character followed by an 'x' followed by two hexadecimal digits. Each constant shall represent a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

For example:

\x63;\xe7;\x63\x68 "\x4d\x61\x79"

5. A character can be represented as a decimal constant. A decimal constant shall be specified as the escape character followed by a 'd' followed by two or three decimal digits. Each constant represents a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

For example:

\d99;\d231;\d99\d104 "\d77\d97\d121"

Implementations may accept single-digit octal, decimal, or hexadecimal constants following the escape character. Only characters existing in the character set for which the locale definition is created shall be specified, whether using symbolic names, the characters themselves, or octal, decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not present in the charmap file can be specified and shall be ignored, as specified under item 1 above.

7.3.1 LC_CTYPE

The *LC_CTYPE* category shall define character classification, case conversion, and other character attributes. In addition, a series of characters can be represented by three adjacent <period> characters representing an ellipsis symbol ("..."). The ellipsis specification shall be interpreted as meaning that all values between the values preceding and following it represent valid characters. The ellipsis specification shall be valid only within a single encoded character set; that is, within a group of characters of the same size. An ellipsis shall be interpreted as including in the list all characters with an encoded value higher than the encoded value of the character preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

For example:

```
\x30;...;\x39;
```

includes in the character class all characters with encoded values between the endpoints.

The following keywords shall be recognized. In the descriptions, the term “automatically included” means that it shall not be an error either to include or omit any of the referenced characters; the implementation provides them if missing (even if the entire keyword is missing) and accepts them silently if present. When the implementation automatically includes a missing character, it shall have an encoded value dependent on the charmap file in effect (see the description of the *localedef* -f option); otherwise, it shall have a value derived from an implementation-defined character mapping.

The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included characters. These only need to be specified if the character values (that is, encoding) differ from the implementation default values. It is not possible to define a locale without these automatically included characters unless some implementation extension is used to prevent their inclusion. Such a definition would not be a proper superset of the C or POSIX locale and, thus, it might not be possible for conforming applications to work properly.

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

upper Define characters to be classified as uppercase letters.

In the POSIX locale, the 26 uppercase letters shall be included:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

In a locale definition file, no character specified for the keywords **cntrl**, **digit**, **punct**, or **space** shall be specified. The uppercase letters <A> to <Z>, as defined in [Section 6.4](#) (on page 129) (the portable character set), are automatically included in this class.

lower Define characters to be classified as lowercase letters.

In the POSIX locale, the 26 lowercase letters shall be included:

a b c d e f g h i j k l m n o p q r s t u v w x y z

In a locale definition file, no character specified for the keywords **cntrl**, **digit**, **punct**, or **space** shall be specified. The lowercase letters <a> to <z> of the portable character set are automatically included in this class.

alpha Define characters to be classified as letters.

In the POSIX locale, all characters in the classes **upper** and **lower** shall be included.

Locale Definition

Locale

4011		In a locale definition file, no character specified for the keywords cntrl , digit ,
4012		punct , or space shall be specified. Characters classified as either upper or
4013		lower are automatically included in this class.
4014	digit	Define the characters to be classified as numeric digits.
4015		In the POSIX locale, only:
4016		0 1 2 3 4 5 6 7 8 9
4017		shall be included.
4018		In a locale definition file, only the digits <zero>, <one>, <two>, <three>,
4019		<four>, <five>, <six>, <seven>, <eight>, and <nine> shall be specified, and in
4020		contiguous ascending sequence by numerical value. The digits <zero> to
4021		<nine> of the portable character set are automatically included in this class.
4022	alnum	Define characters to be classified as letters and numeric digits. Only the
4023		characters specified for the alpha and digit keywords shall be specified.
4024		Characters specified for the keywords alpha and digit are automatically
4025		included in this class.
4026	space	Define characters to be classified as white-space characters.
4027		In the POSIX locale, exactly <space>, <form-feed>, <newline>, <carriage-
4028		return>, <tab>, and <vertical-tab> shall be included.
4029		In a locale definition file, no character specified for the keywords upper ,
4030		lower , alpha , digit , graph , or xdigit shall be specified. The <space>, <form-
4031		feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the portable
4032		character set, and any characters included in the class blank are automatically
4033		included in this class.
4034	cntrl	Define characters to be classified as control characters.
4035		In the POSIX locale, no characters in classes alpha or print shall be included.
4036		In a locale definition file, no character specified for the keywords upper ,
4037		lower , alpha , digit , punct , graph , print , or xdigit shall be specified.
4038	punct	Define characters to be classified as punctuation characters.
4039		In the POSIX locale, neither the <space> nor any characters in classes alpha ,
4040		digit , or cntrl shall be included.
4041		In a locale definition file, no character specified for the keywords upper ,
4042		lower , alpha , digit , cntrl , xdigit , or as the <space> shall be specified.
4043	graph	Define characters to be classified as printable characters, not including the
4044		<space>.
4045		In the POSIX locale, all characters in classes alpha , digit , and punct shall be
4046		included; no characters in class cntrl shall be included.
4047		In a locale definition file, characters specified for the keywords upper , lower ,
4048		alpha , digit , xdigit , and punct are automatically included in this class. No
4049		character specified for the keyword cntrl shall be specified.
4050	print	Define characters to be classified as printable characters, including the
4051		<space>.
4052		In the POSIX locale, all characters in class graph shall be included; no
4053		characters in class cntrl shall be included.

Locale

Locale Definition

4054		In a locale definition file, characters specified for the keywords upper , lower ,
4055		alpha , digit , xdigit , punct , graph , and the <space> are automatically included
4056		in this class. No character specified for the keyword cntrl shall be specified.
4057	xdigit	Define the characters to be classified as hexadecimal digits.
4058		In the POSIX locale, only:
4059		0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
4060		shall be included.
4061		In a locale definition file, only the characters defined for the class digit shall be
4062		specified, in contiguous ascending sequence by numerical value, followed by
4063		one or more sets of six characters representing the hexadecimal digits 10 to 15
4064		inclusive, with each set in ascending order (for example, <A>, , <C>, <D>,
4065		<E>, <F>, <a>, , <c>, <d>, <e>, <f>). The digits <zero> to <nine>, the
4066		uppercase letters <A> to <F>, and the lowercase letters <a> to <f> of the
4067		portable character set are automatically included in this class.
4068	blank	Define characters to be classified as <blank> characters.
4069		In the POSIX locale, only the <space> and <tab> shall be included.
4070		In a locale definition file, the <space> and <tab> are automatically included in
4071		this class.
4072	charclass	Define one or more locale-specific character class names as strings separated
4073		by <semicolon> characters. Each named character class can then be defined
4074		subsequently in the <i>LC_CTYPE</i> definition. A character class name shall consist
4075		of at least one and at most {CHARCLASS_NAME_MAX} bytes of
4076		alphanumeric characters from the portable filename character set. The first
4077		character of a character class name shall not be a digit. The name shall not
4078		match any of the <i>LC_CTYPE</i> keywords defined in this volume of
4079		POSIX.1-2008. Future versions of this standard will not specify any <i>LC_CTYPE</i>
4080		keywords containing uppercase letters.
4081	<i>charclass-name</i>	Define characters to be classified as belonging to the named locale-specific
4082		character class. In the POSIX locale, locale-specific named character classes
4083		need not exist.
4084		If a class name is defined by a charclass keyword, but no characters are
4085		subsequently assigned to it, this is not an error; it represents a class without
4086		any characters belonging to it.
4087		The <i>charclass-name</i> can be used as the <i>property</i> argument to the <i>wctype()</i>
4088		function, in regular expression and shell pattern-matching bracket
4089		expressions, and by the <i>tr</i> command.
4090	toupper	Define the mapping of lowercase letters to uppercase letters.
4091		In the POSIX locale, at a minimum, the 26 lowercase characters:
4092		a b c d e f g h i j k l m n o p q r s t u v w x y z
4093		shall be mapped to the corresponding 26 uppercase characters:
4094		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
4095		In a locale definition file, the operand shall consist of character pairs,
4096		separated by <semicolon> characters. The characters in each character pair
4097		shall be separated by a <comma> and the pair enclosed by parentheses. The

first character in each pair is the lowercase letter, the second the corresponding uppercase letter. Only characters specified for the keywords **lower** and **upper** shall be specified. The lowercase letters <a> to <z>, and their corresponding uppercase letters <A> to <Z>, of the portable character set are automatically included in this mapping, but only when the **toupper** keyword is omitted from the locale definition.

tolower

Define the mapping of uppercase letters to lowercase letters.

In the POSIX locale, at a minimum, the 26 uppercase characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

shall be mapped to the corresponding 26 lowercase characters:

a b c d e f g h i j k l m n o p q r s t u v w x y z

In a locale definition file, the operand shall consist of character pairs, separated by <semicolon> characters. The characters in each character pair shall be separated by a <comma> and the pair enclosed by parentheses. The first character in each pair is the uppercase letter, the second the corresponding lowercase letter. Only characters specified for the keywords **lower** and **upper** shall be specified. If the **tolower** keyword is omitted from the locale definition, the mapping is the reverse mapping of the one specified for **toupper**.

The following table shows the character class combinations allowed:

Table 7-1 Valid Character Class Combinations

In Class	Can Also Belong To										
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
upper	—	—	A	x	x	x	x	A	A	—	x
lower	—	—	A	x	x	x	x	A	A	—	x
alpha	—	—	—	x	x	x	x	A	A	—	x
digit	x	x	x	—	x	x	x	A	A	A	x
space	x	x	x	x	—	—	*	*	*	x	—
cntrl	x	x	x	x	—	—	x	x	x	x	—
punct	x	x	x	x	—	x	—	A	A	x	—
graph	—	—	—	—	—	x	—	—	A	—	—
print	—	—	—	—	—	x	—	—	—	—	—
xdigit	—	—	—	—	x	x	x	A	A	—	x
blank	x	x	x	x	A	—	*	*	*	x	—

Notes:

1. Explanation of codes:

A Automatically included; see text.

— Permitted.

x Mutually-exclusive.

* See note 2.

2. The <space>, which is part of the **space** and **blank** classes, cannot belong to **punct** or **graph**, but shall automatically belong to the **print** class. Other **space** or **blank** characters can be classified as any of **punct**, **graph**, or **print**.

Locale

Locale Definition

4141 7.3.1.1 LC_CTYPE Category in the POSIX Locale

4142 The character classifications for the POSIX locale follow; the code listing depicts the *localedef*
 4143 input, and the table represents the same information, sorted by character.

```

4144 LC_CTYPE
4145 # The following is the POSIX locale LC_CTYPE.
4146 # "alpha" is by default "upper" and "lower"
4147 # "alnum" is by definition "alpha" and "digit"
4148 # "print" is by default "alnum", "punct", and the <space>
4149 # "graph" is by default "alnum" and "punct"
4150 #
4151 upper    <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
4152          <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>;\
4153 #
4154 lower    <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
4155          <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
4156 #
4157 digit    <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
4158          <seven>;<eight>;<nine>
4159 #
4160 space    <tab>;<newline>;<vertical-tab>;<form-feed>;\
4161          <carriage-return>;<space>
4162 #
4163 cntrl    <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
4164          <form-feed>;<carriage-return>;\
4165          <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
4166          <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
4167          <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
4168          <IS1>;<DEL>
4169 #
4170 punct    <exclamation-mark>;<quotation-mark>;<number-sign>;\
4171          <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
4172          <left-parenthesis>;<right-parenthesis>;<asterisk>;\
4173          <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
4174          <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
4175          <greater-than-sign>;<question-mark>;<commercial-at>;\
4176          <left-square-bracket>;<backslash>;<right-square-bracket>;\
4177          <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
4178          <vertical-line>;<right-curly-bracket>;<tilde>
4179 #
4180 xdigit    <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
4181          <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
4182 #
4183 blank    <space>;<tab>
4184 #
4185 toupper  (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
4186          (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
4187          (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
4188          (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
4189          (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);(<z>,<Z>)
4190 #
4191 tolower  (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);\
4192          (<F>,<f>);(<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);\

```

Locale Definition

Locale

```

4193         (<K>, <k>); (<L>, <l>); (<M>, <m>); (<N>, <n>); (<O>, <o>); \
4194         (<P>, <p>); (<Q>, <q>); (<R>, <r>); (<S>, <s>); (<T>, <t>); \
4195         (<U>, <u>); (<V>, <v>); (<W>, <w>); (<X>, <x>); (<Y>, <y>); (<Z>, <z>)
4196     END LC_CTYPE

```

Symbolic Name	Other Case	Character Classes
<NUL>		cntrl
<SOH>		cntrl
<STX>		cntrl
<ETX>		cntrl
<EOT>		cntrl
<ENQ>		cntrl
<ACK>		cntrl
<alert>		cntrl
<backspace>		cntrl
<tab>		cntrl, space, blank
<newline>		cntrl, space
<vertical-tab>		cntrl, space
<form-feed>		cntrl, space
<carriage-return>		cntrl, space
<SO>		cntrl
<SI>		cntrl
<DLE>		cntrl
<DC1>		cntrl
<DC2>		cntrl
<DC3>		cntrl
<DC4>		cntrl
<NAK>		cntrl
<SYN>		cntrl
<ETB>		cntrl
<CAN>		cntrl
		cntrl
<SUB>		cntrl
<ESC>		cntrl
<IS4>		cntrl
<IS3>		cntrl
<IS2>		cntrl
<IS1>		cntrl
<space>		space, print, blank
<exclamation-mark>		punct, print, graph
<quotation-mark>		punct, print, graph
<number-sign>		punct, print, graph
<dollar-sign>		punct, print, graph
<percent-sign>		punct, print, graph
<ampersand>		punct, print, graph
<apostrophe>		punct, print, graph
<left-parenthesis>		punct, print, graph
<right-parenthesis>		punct, print, graph
<asterisk>		punct, print, graph
<plus-sign>		punct, print, graph
<comma>		punct, print, graph
<hyphen>		punct, print, graph

Locale

Locale Definition

	Symbolic Name	Other Case	Character Classes
4244	<period>		punct, print, graph
4245	<slash>		punct, print, graph
4246	<zero>		digit, xdigit, print, graph
4247	<one>		digit, xdigit, print, graph
4248	<two>		digit, xdigit, print, graph
4249	<three>		digit, xdigit, print, graph
4250	<four>		digit, xdigit, print, graph
4251	<five>		digit, xdigit, print, graph
4252	<six>		digit, xdigit, print, graph
4253	<seven>		digit, xdigit, print, graph
4254	<eight>		digit, xdigit, print, graph
4255	<nine>		digit, xdigit, print, graph
4256	<colon>		punct, print, graph
4257	<semicolon>		punct, print, graph
4258	<less-than-sign>		punct, print, graph
4259	<equals-sign>		punct, print, graph
4260	<greater-than-sign>		punct, print, graph
4261	<question-mark>		punct, print, graph
4262	<commercial-at>		punct, print, graph
4263	<A>	<a>	upper, xdigit, alpha, print, graph
4264			upper, xdigit, alpha, print, graph
4265	<C>	<c>	upper, xdigit, alpha, print, graph
4266	<D>	<d>	upper, xdigit, alpha, print, graph
4267	<E>	<e>	upper, xdigit, alpha, print, graph
4268	<F>	<f>	upper, xdigit, alpha, print, graph
4269	<G>	<g>	upper, alpha, print, graph
4270	<H>	<h>	upper, alpha, print, graph
4271	<I>	<i>	upper, alpha, print, graph
4272	<J>	<j>	upper, alpha, print, graph
4273	<K>	<k>	upper, alpha, print, graph
4274	<L>	<l>	upper, alpha, print, graph
4275	<M>	<m>	upper, alpha, print, graph
4276	<N>	<n>	upper, alpha, print, graph
4277	<O>	<o>	upper, alpha, print, graph
4278	<P>	<p>	upper, alpha, print, graph
4279	<Q>	<q>	upper, alpha, print, graph
4280	<R>	<r>	upper, alpha, print, graph
4281	<S>	<s>	upper, alpha, print, graph
4282	<T>	<t>	upper, alpha, print, graph
4283	<U>	<u>	upper, alpha, print, graph
4284	<V>	<v>	upper, alpha, print, graph
4285	<W>	<w>	upper, alpha, print, graph
4286	<X>	<x>	upper, alpha, print, graph
4287	<Y>	<y>	upper, alpha, print, graph
4288	<Z>	<z>	upper, alpha, print, graph
4289	<left-square-bracket>		punct, print, graph
4290	<backslash>		punct, print, graph
4291	<right-square-bracket>		punct, print, graph
4292	<circumflex>		punct, print, graph
4293	<underscore>		punct, print, graph
4294	<grave-accent>		punct, print, graph
4295			

	Symbolic Name	Other Case	Character Classes
4296			
4297	<a>	<A>	lower, xdigit, alpha, print, graph
4298			lower, xdigit, alpha, print, graph
4299	<c>	<C>	lower, xdigit, alpha, print, graph
4300	<d>	<D>	lower, xdigit, alpha, print, graph
4301	<e>	<E>	lower, xdigit, alpha, print, graph
4302	<f>	<F>	lower, xdigit, alpha, print, graph
4303	<g>	<G>	lower, alpha, print, graph
4304	<h>	<H>	lower, alpha, print, graph
4305	<i>	<I>	lower, alpha, print, graph
4306	<j>	<J>	lower, alpha, print, graph
4307	<k>	<K>	lower, alpha, print, graph
4308	<l>	<L>	lower, alpha, print, graph
4309	<m>	<M>	lower, alpha, print, graph
4310	<n>	<N>	lower, alpha, print, graph
4311	<o>	<O>	lower, alpha, print, graph
4312	<p>	<P>	lower, alpha, print, graph
4313	<q>	<Q>	lower, alpha, print, graph
4314	<r>	<R>	lower, alpha, print, graph
4315	<s>	<S>	lower, alpha, print, graph
4316	<t>	<T>	lower, alpha, print, graph
4317	<u>	<U>	lower, alpha, print, graph
4318	<v>	<V>	lower, alpha, print, graph
4319	<w>	<W>	lower, alpha, print, graph
4320	<x>	<X>	lower, alpha, print, graph
4321	<y>	<Y>	lower, alpha, print, graph
4322	<z>	<Z>	lower, alpha, print, graph
4323	<left-curly-bracket>		punct, print, graph
4324	<vertical-line>		punct, print, graph
4325	<right-curly-bracket>		punct, print, graph
4326	<tilde>		punct, print, graph
4327	<DEL>		cntrl

7.3.2 LC_COLLATE

The *LC_COLLATE* category provides a collation sequence definition for numerous utilities in the Shell and Utilities volume of POSIX.1-2008 (*sort*, *uniq*, and so on), regular expression matching (see Chapter 9, on page 181), and the *strcoll()*, *strxfrm()*, *wscoll()*, and *wcsxfrm()* functions in the System Interfaces volume of POSIX.1-2008.

A collation sequence definition shall define the relative order between collating elements (characters and multi-character collating elements) in the locale. This order is expressed in terms of collation values; that is, by assigning each element one or more collation values (also known as collation weights). This does not imply that implementations shall assign such values, but that ordering of strings using the resultant collation definition in the locale behaves as if such assignment is done and used in the collation process. At least the following capabilities are provided:

1. **Multi-character collating elements.** Specification of multi-character collating elements (that is, sequences of two or more characters to be collated as an entity).

Locale

Locale Definition

2. **User-defined ordering of collating elements.** Each collating element shall be assigned a collation value defining its order in the character (or basic) collation sequence. This ordering is used by regular expressions and pattern matching and, unless collation weights are explicitly specified, also as the collation weight to be used in sorting.
3. **Multiple weights and equivalence classes.** Collating elements can be assigned one or more (up to the limit {COLL_WEIGHTS_MAX}, as defined in [<limits.h>](#)) collating weights for use in sorting. The first weight is hereafter referred to as the primary weight.
4. **One-to-many mapping.** A single character is mapped into a string of collating elements.
5. **Equivalence class definition.** Two or more collating elements have the same collation value (primary weight).
6. **Ordering by weights.** When two strings are compared to determine their relative order, the two strings are first broken up into a series of collating elements; the elements in each successive pair of elements are then compared according to the relative primary weights for the elements. If equal, and more than one weight has been assigned, then the pairs of collating elements are re-compared according to the relative subsequent weights, until either a pair of collating elements compare unequal or the weights are exhausted.

The following keywords shall be recognized in a collation sequence definition. They are described in detail in the following sections.

copy	Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.
collating-element	Define a collating-element symbol representing a multi-character collating element. This keyword is optional.
collating-symbol	Define a collating symbol for use in collation order statements. This keyword is optional.
order_start	Define collation rules. This statement shall be followed by one or more collation order statements, assigning character collation values and collation weights to collating elements.
order_end	Specify the end of the collation-order statements.

7.3.2.1 The collating-element Keyword

In addition to the collating elements in the character set, the **collating-element** keyword can be used to define multi-character collating elements. The syntax is as follows:

```
"collating-element %s from \"%s\"\\n", <collating-symbol>, <string>
```

The **<collating-symbol>** operand shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A **<collating-element>** defined via this keyword is only recognized with the **LC_COLLATE** category.

For example:

```
collating-element <ch> from "<c><h>"
collating-element <e-acute> from "<acute><e>"
collating-element <ll> from "ll"
```


Locale Definition

Locale

4384 7.3.2.2 The collating-symbol Keyword

4385 This keyword shall be used to define symbols for use in collation sequence statements; that is,
4386 between the **order_start** and the **order_end** keywords. The syntax is as follows:

4387 "collating-symbol %s\n", <collating-symbol>

4388 The <collating-symbol> shall be a symbolic name, enclosed between angle brackets ('<' and
4389 '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any
4390 other symbolic name defined in this collation definition. A <collating-symbol> defined via this
4391 keyword is only recognized within the LC_COLLATE category.

4392 For example:

4393 collating-symbol <UPPER_CASE>

4394 collating-symbol <HIGH>

4395 The **collating-symbol** keyword defines a symbolic name that can be associated with a relative
4396 position in the character order sequence. While such a symbolic name does not represent any
4397 collating element, it can be used as a weight.

4398 7.3.2.3 The order_start Keyword

4399 The **order_start** keyword shall precede collation order entries and also define the number of
4400 weights for this collation sequence definition and other collation rules. The syntax is as follows:

4401 "order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...

4402 The operands to the **order_start** keyword are optional. If present, the operands define rules to be
4403 applied when strings are compared. The number of operands define how many weights each
4404 element is assigned; if no operands are present, one **forward** operand is assumed. If present, the
4405 first operand defines rules to be applied when comparing strings using the first (primary)
4406 weight; the second when comparing strings using the second weight, and so on. Operands shall
4407 be separated by <semicolon> characters (';'). Each operand shall consist of one or more
4408 collation directives, separated by <comma> characters (','). If the number of operands exceeds
4409 the {COLL_WEIGHTS_MAX} limit, the utility shall issue a warning message. The following
4410 directives shall be supported:

4411 **forward** Specifies that comparison operations for the weight level shall proceed from start
4412 of string towards the end of string.

4413 **backward** Specifies that comparison operations for the weight level shall proceed from end of
4414 string towards the beginning of string.

4415 **position** Specifies that comparison operations for the weight level shall consider the relative
4416 position of elements in the strings not subject to **IGNORE**. The string containing
4417 an element not subject to **IGNORE** after the fewest collating elements subject to
4418 **IGNORE** from the start of the compare shall collate first. If both strings contain a
4419 character not subject to **IGNORE** in the same relative position, the collating values
4420 assigned to the elements shall determine the ordering. In case of equality,
4421 subsequent characters not subject to **IGNORE** shall be considered in the same
4422 manner.

4423 The directives **forward** and **backward** are mutually-exclusive.

4424 If no operands are specified, a single **forward** operand shall be assumed.

Locale

Locale Definition

For example:

```
order_start    forward;backward
```

7.3.2.4 Collation Order

The **order_start** keyword shall be followed by collating identifier entries. The syntax for the collating element entries is as follows:

```
"%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...
```

Each *collating-identifier* shall consist of either a character (in any of the forms defined in [Section 7.3](#), on page 136), a *<collating-element>*, a *<collating-symbol>*, an ellipsis, or the special symbol **UNDEFINED**. The order in which collating elements are specified determines the character order sequence, such that each collating element shall compare less than the elements following it.

A *<collating-element>* shall be used to specify multi-character collating elements, and indicates that the character sequence specified via the *<collating-element>* is to be collated as a unit and in the relative order specified by its place.

A *<collating-symbol>* can be used to define a position in the relative order for use in weights. No weights shall be specified with a *<collating-symbol>*.

The ellipsis symbol specifies that a sequence of characters shall collate according to their encoded character values. It shall be interpreted as indicating that all characters with a coded character set value higher than the value of the character in the preceding line, and lower than the coded character set value for the character in the following line, in the current coded character set, shall be placed in the character collation order between the previous and the following character in ascending order according to their coded character set values. An initial ellipsis shall be interpreted as if the preceding line specified the NUL character, and a trailing ellipsis as if the following line specified the highest coded character set value in the current coded character set. An ellipsis shall be treated as invalid if the preceding or following lines do not specify characters in the current coded character set. The use of the ellipsis symbol ties the definition to a specific coded character set and may preclude the definition from being portable between implementations.

The symbol **UNDEFINED** shall be interpreted as including all coded character set values not specified explicitly or via the ellipsis symbol. Such characters shall be inserted in the character collation order at the point indicated by the symbol, and in ascending order according to their coded character set values. If no **UNDEFINED** symbol is specified, and the current coded character set contains characters not specified in this section, the utility shall issue a warning message and place such characters at the end of the character collation order.

The optional operands for each collation-element shall be used to define the primary, secondary, or subsequent weights for the collating element. The first operand specifies the relative primary weight, the second the relative secondary weight, and so on. Two or more collation-elements can be assigned the same weight; they belong to the same "equivalence class" if they have the same primary weight. Collation shall behave as if, for each weight level, elements subject to **IGNORE** are removed, unless the **position** collation directive is specified for the corresponding level with the **order_start** keyword. Then each successive pair of elements shall be compared according to the relative weights for the elements. If the two strings compare equal, the process shall be repeated for the next weight level, up to the limit {COLL_WEIGHTS_MAX}.

Weights shall be expressed as characters (in any of the forms specified in [Section 7.3](#), on page 136), *<collating-symbol>*s, *<collating-element>*s, an ellipsis, or the special symbol **IGNORE**. A single character, a *<collating-symbol>*, or a *<collating-element>* shall represent the relative position

in the character collating sequence of the character or symbol, rather than the character or characters themselves. Thus, rather than assigning absolute values to weights, a particular weight is expressed using the relative order value assigned to a collating element based on its order in the character collation sequence.

One-to-many mapping is indicated by specifying two or more concatenated characters or symbolic names. For example, if the <eszet> is given the string "<s><s>" as a weight, comparisons are performed as if all occurrences of the <eszet> are replaced by "<s><s>" (assuming that "<s>" has the collating weight "<s>"). If it is necessary to define <eszet> and "<s><s>" as an equivalence class, then a collating element must be defined for the string "ss".

All characters specified via an ellipsis shall by default be assigned unique weights, equal to the relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special symbol shall by default be assigned the same primary weight (that is, they belong to the same equivalence class). An ellipsis symbol as a weight shall be interpreted to mean that each character in the sequence shall have unique weights, equal to the relative order of their character in the character collation sequence. The use of the ellipsis as a weight shall be treated as an error if the collating element is neither an ellipsis nor the special symbol **UNDEFINED**.

The special keyword **IGNORE** as a weight shall indicate that when strings are compared using the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that is, as if the string did not contain the collating element. In regular expressions and pattern matching, all characters that are subject to **IGNORE** in their primary weight form an equivalence class.

An empty operand shall be interpreted as the collating element itself.

For example, the order statement:

```
<a>      <a>;<a>
```

is equal to:

```
<a>
```

An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be interpreted as the value of each character defined by the ellipsis.

The collation order as defined in this section affects the interpretation of bracket expressions in regular expressions (see [Section 9.3.5](#), on page 184).

For example:

```
order_start forward;backward
UNDEFINED  IGNORE;IGNORE
<LOW>
<space>    <LOW>;<space>
...        <LOW>;...
<a>        <a>;<a>
<a-acute>  <a>;<a-acute>
<a-grave>  <a>;<a-grave>
<A>        <a>;<A>
<A-acute>  <a>;<A-acute>
<A-grave>  <a>;<A-grave>
<ch>       <ch>;<ch>
<Ch>       <ch>;<Ch>
<s>        <s>;<s>
<eszet>    "<s><s>"; "<eszet><eszet>"
order_end
```

*Locale**Locale Definition*

This example is interpreted as follows:

1. The **UNDEFINED** means that all characters not specified in this definition (explicitly or via the ellipsis) shall be ignored for collation purposes.
2. All characters between <space> and 'a' shall have the same primary equivalence class and individual secondary weights based on their ordinal encoded values.
3. All characters based on the uppercase or lowercase character 'a' belong to the same primary equivalence class.
4. The multi-character collating element <ch> is represented by the collating symbol <ch> and belongs to the same primary equivalence class as the multi-character collating element <Ch>.

7.3.2.5 *The order_end Keyword*

The collating order entries shall be terminated with an **order_end** keyword.

7.3.2.6 *LC_COLLATE Category in the POSIX Locale*

The collation sequence definition of the POSIX locale follows; the code listing depicts the *localedef* input.

```
LC_COLLATE
# This is the POSIX locale definition for the LC_COLLATE category.
# The order is the same as in the ASCII codeset.
order_start forward
<NUL>
<SOH>
<STX>
<ETX>
<EOT>
<ENQ>
<ACK>
<alert>
<backspace>
<tab>
<newline>
<vertical-tab>
<form-feed>
<carriage-return>
<SO>
<SI>
<DLE>
<DC1>
<DC2>
<DC3>
<DC4>
<NAK>
<SYN>
<ETB>
<CAN>
<EM>
<SUB>
```

*Locale Definition**Locale*

4564	<ESC>
4565	<IS4>
4566	<IS3>
4567	<IS2>
4568	<IS1>
4569	<space>
4570	<exclamation-mark>
4571	<quotation-mark>
4572	<number-sign>
4573	<dollar-sign>
4574	<percent-sign>
4575	<ampersand>
4576	<apostrophe>
4577	<left-parenthesis>
4578	<right-parenthesis>
4579	<asterisk>
4580	<plus-sign>
4581	<comma>
4582	<hyphen>
4583	<period>
4584	<slash>
4585	<zero>
4586	<one>
4587	<two>
4588	<three>
4589	<four>
4590	<five>
4591	<six>
4592	<seven>
4593	<eight>
4594	<nine>
4595	<colon>
4596	<semicolon>
4597	<less-than-sign>
4598	<equals-sign>
4599	<greater-than-sign>
4600	<question-mark>
4601	<commercial-at>
4602	<A>
4603	
4604	<C>
4605	<D>
4606	<E>
4607	<F>
4608	<G>
4609	<H>
4610	<I>
4611	<J>
4612	<K>
4613	<L>
4614	<M>
4615	<N>
4616	<O>

*Locale**Locale Definition*

4617 <P>
 4618 <Q>
 4619 <R>
 4620 <S>
 4621 <T>
 4622 <U>
 4623 <V>
 4624 <W>
 4625 <X>
 4626 <Y>
 4627 <Z>
 4628 <left-square-bracket>
 4629 <backslash>
 4630 <right-square-bracket>
 4631 <circumflex>
 4632 <underscore>
 4633 <grave-accent>
 4634 <a>
 4635
 4636 <c>
 4637 <d>
 4638 <e>
 4639 <f>
 4640 <g>
 4641 <h>
 4642 <i>
 4643 <j>
 4644 <k>
 4645 <l>
 4646 <m>
 4647 <n>
 4648 <o>
 4649 <p>
 4650 <q>
 4651 <r>
 4652 <s>
 4653 <t>
 4654 <u>
 4655 <v>
 4656 <w>
 4657 <x>
 4658 <y>
 4659 <z>
 4660 <left-curly-bracket>
 4661 <vertical-line>
 4662 <right-curly-bracket>
 4663 <tilde>
 4664 <DEL>
 4665 order_end
 4666 #
 4667 END LC_COLLATE

7.3.3 LC_MONETARY

The *LC_MONETARY* category shall define the rules and symbols that are used to format monetary numeric information.

This information is available through the *localeconv()* function and is used by the *strfmon()* function.

Some of the information is also available in an alternative form via the *nl_langinfo()* function (see CRNCYSTR in [<langinfo.h>](#)).

The following items are defined in this category of the locale. The item names are the keywords recognized by the *localedef* utility when defining a locale. They are also similar to the member names of the *lconv* structure defined in [<locale.h>](#); see [<locale.h>](#) for the exact symbols in the header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the empty string (" ") for unspecified or size zero string items.

In a locale definition file, the operands are strings, formatted as indicated by the grammar in [Section 7.4](#) (on page 165). For some keywords, the strings can contain only integers. Keywords that are not provided, string values set to the empty string (" "), or integer keywords set to -1, are used to indicate that the value is not available in the locale. The following keywords shall be recognized:

copy	Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.
	Note: This is a <i>localedef</i> utility keyword, unavailable through <i>localeconv()</i> .
int_curr_symbol	The international currency symbol. The operand shall be a four-character string, with the first three characters containing the alphabetic international currency symbol. The international currency symbol should be chosen in accordance with those specified in the ISO 4217 standard. The fourth character shall be the character used to separate the international currency symbol from the monetary quantity.
currency_symbol	The string that shall be used as the local currency symbol.
mon_decimal_point	The operand is a string containing the symbol that shall be used as the decimal delimiter (radix character) in monetary formatted quantities.
mon_thousands_sep	The operand is a string containing the symbol that shall be used as a separator for groups of digits to the left of the decimal delimiter in formatted monetary quantities.
mon_grouping	Define the size of each group of digits in formatted monetary quantities. The operand is a sequence of integers separated by <semicolon> characters. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not -1, then the size of the previous group (if any) shall be repeatedly used for the remainder of the digits. If the last integer is -1, then no further grouping shall be performed.
positive_sign	A string that shall be used to indicate a non-negative-valued formatted monetary quantity.
negative_sign	A string that shall be used to indicate a negative-valued formatted monetary quantity.

Locale

Locale Definition

4713	int_frac_digits	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using int_curr_symbol .
4714		
4715		
4716	frac_digits	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using currency_symbol .
4717		
4718		
4719	p_cs_precedes	An integer set to 1 if the currency_symbol precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
4720		
4721		
4722	p_sep_by_space	Set to a value indicating the separation of the currency_symbol , the sign string, and the value for a non-negative formatted monetary quantity.
4723		
4724		The values of p_sep_by_space , n_sep_by_space , int_p_sep_by_space , and int_n_sep_by_space are interpreted according to the following:
4725		
4726		0 No <space> separates the currency symbol and value.
4727		1 If the currency symbol and sign string are adjacent, a <space> separates them from the value; otherwise, a <space> separates the currency symbol from the value.
4728		
4729		
4730		2 If the currency symbol and sign string are adjacent, a <space> separates them; otherwise, a <space> separates the sign string from the value.
4731		
4732		
4733	n_cs_precedes	An integer set to 1 if the currency_symbol precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4734		
4735		
4736	n_sep_by_space	Set to a value indicating the separation of the currency_symbol , the sign string, and the value for a negative formatted monetary quantity.
4737		
4738	p_sign_posn	An integer set to a value indicating the positioning of the positive_sign for a monetary quantity with a non-negative value. The following integer values shall be recognized for int_n_sign_posn , int_p_sign_posn , n_sign_posn , and p_sign_posn :
4739		
4740		
4741		
4742		0 Parentheses enclose the quantity and the currency_symbol .
4743		1 The sign string precedes the quantity and the currency_symbol .
4744		2 The sign string succeeds the quantity and the currency_symbol .
4745		3 The sign string precedes the currency_symbol .
4746		4 The sign string succeeds the currency_symbol .
4747	n_sign_posn	An integer set to a value indicating the positioning of the negative_sign for a negative formatted monetary quantity.
4748		
4749	int_p_cs_precedes	An integer set to 1 if the int_curr_symbol precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
4750		
4751		
4752	int_n_cs_precedes	An integer set to 1 if the int_curr_symbol precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4753		
4754		

Locale Definition

Locale

4755	int_p_sep_by_space	Set to a value indicating the separation of the int_curr_symbol , the sign string, and the value for a non-negative internationally formatted monetary quantity.
4756		
4757		
4758	int_n_sep_by_space	Set to a value indicating the separation of the int_curr_symbol , the sign string, and the value for a negative internationally formatted monetary quantity.
4759		
4760		
4761	int_p_sign_posn	An integer set to a value indicating the positioning of the positive_sign for a positive monetary quantity formatted with the international format.
4762		
4763	int_n_sign_posn	An integer set to a value indicating the positioning of the negative_sign for a negative monetary quantity formatted with the international format.
4764		

4765 7.3.3.1 LC_MONETARY Category in the POSIX Locale

4766 The monetary formatting definitions for the POSIX locale follow; the code listing depicting the
 4767 *localedef* input, the table representing the same information with the addition of *localeconv()* and
 4768 *nl_langinfo()* formats. All values are unspecified in the POSIX locale.

```

4769 LC_MONETARY
4770 # This is the POSIX locale definition for
4771 # the LC_MONETARY category.
4772 #
4773 int_curr_symbol      " "
4774 currency_symbol      " "
4775 mon_decimal_point    " "
4776 mon_thousands_sep   " "
4777 mon_grouping         -1
4778 positive_sign        " "
4779 negative_sign        " "
4780 int_frac_digits      -1
4781 frac_digits          -1
4782 p_cs_precedes        +1
4783 p_sep_by_space       -1
4784 n_cs_precedes        -1
4785 n_sep_by_space       -1
4786 p_sign_posn          -1
4787 n_sign_posn          -1
4788 int_p_cs_precedes    -1
4789 int_p_sep_by_space   -1
4790 int_n_cs_precedes    -1
4791 int_n_sep_by_space   -1
4792 int_p_sign_posn      -1
4793 int_n_sign_posn      -1
4794 #
4795 END LC_MONETARY

```


Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
int_curr_symbol	—	N/A	" "	" "
currency_symbol	CRNCYSTR	N/A	" "	" "
mon_decimal_point	—	N/A	" "	" "
mon_thousands_sep	—	N/A	" "	" "
mon_grouping	—	N/A	" "	–1
positive_sign	—	N/A	" "	" "
negative_sign	—	N/A	" "	" "
int_frac_digits	—	N/A	{CHAR_MAX}	–1
frac_digits	—	N/A	{CHAR_MAX}	–1
p_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	–1
p_sep_by_space	—	N/A	{CHAR_MAX}	–1
n_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	–1
n_sep_by_space	—	N/A	{CHAR_MAX}	–1
p_sign_posn	—	N/A	{CHAR_MAX}	–1
n_sign_posn	—	N/A	{CHAR_MAX}	–1
int_p_cs_precedes	—	N/A	{CHAR_MAX}	–1
int_p_sep_by_space	—	N/A	{CHAR_MAX}	–1
int_n_cs_precedes	—	N/A	{CHAR_MAX}	–1
int_n_sep_by_space	—	N/A	{CHAR_MAX}	–1
int_p_sign_posn	—	N/A	{CHAR_MAX}	–1
int_n_sign_posn	—	N/A	{CHAR_MAX}	–1

The entry N/A indicates that the value is not available in the POSIX locale.

7.3.4 LC_NUMERIC

The *LC_NUMERIC* category shall define the rules and symbols that are used to format non-monetary numeric information. This information is available through the *localeconv()* function.

Some of the information is also available in an alternative form via the *nl_langinfo()* function.

The following items are defined in this category of the locale. The item names are the keywords recognized by the *localedef* utility when defining a locale. They are also similar to the member names of the *lconv* structure defined in [<locale.h>](#); see [<locale.h>](#) for the exact symbols in the header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the empty string (" ") for unspecified or size zero string items.

In a locale definition file, the operands are strings, formatted as indicated by the grammar in [Section 7.4](#) (on page 165). For some keywords, the strings can only contain integers. Keywords that are not provided, string values set to the empty string (" "), or integer keywords set to –1, shall be used to indicate that the value is not available in the locale. The following keywords shall be recognized:

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

Note: This is a *localedef* utility keyword, unavailable through *localeconv()*.

decimal_point The operand is a string containing the symbol that shall be used as the decimal delimiter (radix character) in numeric, non-monetary formatted quantities. This keyword cannot be omitted and cannot be set to the empty string. In contexts where standards limit the **decimal_point** to a single byte, the result of specifying a multi-byte operand shall be unspecified.

thousands_sep The operand is a string containing the symbol that shall be used as a separator for groups of digits to the left of the decimal delimiter in numeric, non-monetary formatted monetary quantities. In contexts where standards limit the **thousands_sep** to a single byte, the result of specifying a multi-byte operand shall be unspecified.

grouping Define the size of each group of digits in formatted non-monetary quantities. The operand is a sequence of integers separated by <semicolon> characters. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not -1, then the size of the previous group (if any) shall be repeatedly used for the remainder of the digits. If the last integer is -1, then no further grouping shall be performed.

7.3.4.1 LC_NUMERIC Category in the POSIX Locale

The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing depicting the *localedef* input, the table representing the same information with the addition of *localeconv()* values, and *nl_langinfo()* constants.

```
LC_NUMERIC
# This is the POSIX locale definition for
# the LC_NUMERIC category.
#
decimal_point      "<period>"
thousands_sep      ""
grouping           -1
#
END LC_NUMERIC
```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
decimal_point	RADIXCHAR	."	."	.
thousands_sep	THOUSEP	N/A	""	""
grouping	—	N/A	""	-1

The entry N/A indicates that the value is not available in the POSIX locale.

7.3.5 LC_TIME

The *LC_TIME* category shall define the interpretation of the conversion specifications supported by the *date* utility and shall affect the behavior of the *strftime()*, *wcsftime()*, *strptime()*, and *nl_langinfo()* functions. Since the interfaces for C-language access and locale definition differ significantly, they are described separately.

7.3.5.1 LC_TIME Locale Definition

In a locale definition, the following mandatory keywords shall be recognized:

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

Locale

Locale Definition

4883	abday	Define the abbreviated weekday names, corresponding to the %a conversion specification (conversion specification in the <i>strftime()</i> , <i>wcsftime()</i> , and <i>strptime()</i> functions). The operand shall consist of seven &ltsemicolon>-separated strings, each surrounded by double-quotes. The first string shall be the abbreviated name of the day corresponding to Sunday, the second the abbreviated name of the day corresponding to Monday, and so on.
4884		
4885		
4886		
4887		
4888		
4889	day	Define the full weekday names, corresponding to the %A conversion specification. The operand shall consist of seven &ltsemicolon>-separated strings, each surrounded by double-quotes. The first string is the full name of the day corresponding to Sunday, the second the full name of the day corresponding to Monday, and so on.
4890		
4891		
4892		
4893		
4894	abmon	Define the abbreviated month names, corresponding to the %b conversion specification. The operand shall consist of twelve &ltsemicolon>-separated strings, each surrounded by double-quotes. The first string shall be the abbreviated name of the first month of the year (January), the second the abbreviated name of the second month, and so on.
4895		
4896		
4897		
4898		
4899	mon	Define the full month names, corresponding to the %B conversion specification. The operand shall consist of twelve &ltsemicolon>-separated strings, each surrounded by double-quotes. The first string shall be the full name of the first month of the year (January), the second the full name of the second month, and so on.
4900		
4901		
4902		
4903		
4904	d_t_fmt	Define the appropriate date and time representation, corresponding to the %c conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in the table in Table 5-1 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').
4905		
4906		
4907		
4908		
4909	d_fmt	Define the appropriate date representation, corresponding to the %x conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in Table 5-1 (on page 121).
4910		
4911		
4912		
4913	t_fmt	Define the appropriate time representation, corresponding to the %X conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in Table 5-1 (on page 121).
4914		
4915		
4916		
4917	am_pm	Define the appropriate representation of the <i>ante-meridiem</i> and <i>post-meridiem</i> strings, corresponding to the %p conversion specification. The operand shall consist of two strings, separated by a &ltsemicolon>, each surrounded by double-quotes. The first string shall represent the <i>ante-meridiem</i> designation, the last string the <i>post-meridiem</i> designation.
4918		
4919		
4920		
4921		
4922	t_fmt_ampm	Define the appropriate time representation in the 12-hour clock format with am_pm , corresponding to the %r conversion specification. The operand shall consist of a string and can contain any combination of characters and conversion specifications. If the string is empty, the 12-hour format is not supported in the locale.
4923		
4924		
4925		
4926		
4927	era	Define how years are counted and displayed for each era in a locale. The operand shall consist of &ltsemicolon>-separated strings. Each string shall be an era description segment with the format:
4928		
4929		
4930		<i>direction:offset:start_date:end_date:era_name:era_format</i>

Locale Definition

Locale

4931		according to the definitions below. There can be as many era description
4932		segments as are necessary to describe the different eras.
4933	Note:	The start of an era might not be the earliest point in the era—it may be the
4934		latest. For example, the Christian era BC starts on the day before January 1,
4935		AD 1, and increases with earlier time.
4936	<i>direction</i>	Either a '+' or a '-' character. The '+' character shall indicate
4937		that years closer to the <i>start_date</i> have lower numbers than those
4938		closer to the <i>end_date</i> . The '-' character shall indicate that years
4939		closer to the <i>start_date</i> have higher numbers than those closer to
4940		the <i>end_date</i> .
4941	<i>offset</i>	The number of the year closest to the <i>start_date</i> in the era,
4942		corresponding to the %EY conversion specification.
4943	<i>start_date</i>	A date in the form <i>yyyy/mm/dd</i> , where <i>yyyy</i> , <i>mm</i> , and <i>dd</i> are the
4944		year, month, and day numbers respectively of the start of the era.
4945		Years prior to AD 1 shall be represented as negative numbers.
4946	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> ,
4947		or one of the two special values "*" or "+*". The value "*" shall
4948		indicate that the ending date is the beginning of time. The
4949		value "+*" shall indicate that the ending date is the end of time.
4950	<i>era_name</i>	A string representing the name of the era, corresponding to the
4951		%EC conversion specification.
4952	<i>era_format</i>	A string for formatting the year in the era, corresponding to the
4953		%EY conversion specification.
4954	era_d_fmt	Define the format of the date in alternative era notation, corresponding to the
4955		%Ex conversion specification.
4956	era_t_fmt	Define the locale's appropriate alternative time format, corresponding to the
4957		%EX conversion specification.
4958	era_d_t_fmt	Define the locale's appropriate alternative date and time format,
4959		corresponding to the %Ec conversion specification.
4960	alt_digits	Define alternative symbols for digits, corresponding to the %O modified
4961		conversion specification. The operand shall consist of <semicolon>-separated
4962		strings, each surrounded by double-quotes. The first string shall be the
4963		alternative symbol corresponding with zero, the second string the symbol
4964		corresponding with one, and so on. Up to 100 alternative symbol strings can
4965		be specified. The %O modifier shall indicate that the string corresponding to
4966		the value specified via the conversion specification shall be used instead of the
4967		value.
4968	7.3.5.2	<i>LC_TIME C-Language Access</i>
4969		The following constants used to identify items of <i>langinfo</i> data can be used as arguments to the
4970		<i>nl_langinfo()</i> function to access information in the <i>LC_TIME</i> category. These constants are
4971		defined in the <langinfo.h> header.
4972	ABDAY_x	The abbreviated weekday names (for example, Sun), where <i>x</i> is a number
4973		from 1 to 7.

Locale

Locale Definition

4974	DAY_ <i>x</i>	The full weekday names (for example, Sunday), where <i>x</i> is a number from 1 to 7.
4975		
4976	ABMON_ <i>x</i>	The abbreviated month names (for example, Jan), where <i>x</i> is a number from 1 to 12.
4977		
4978	MON_ <i>x</i>	The full month names (for example, January), where <i>x</i> is a number from 1 to 12.
4979		
4980	D_T_FMT	The appropriate date and time representation.
4981	D_FMT	The appropriate date representation.
4982	T_FMT	The appropriate time representation.
4983	AM_STR	The appropriate ante-meridiem affix.
4984	PM_STR	The appropriate post-meridiem affix.
4985	T_FMT_AMPM	The appropriate time representation in the 12-hour clock format with AM_STR and PM_STR.
4986		
4987	ERA	The era description segments, which describe how years are counted and displayed for each era in a locale. Each era description segment shall have the format:
4988		
4989		
4990		<i>direction:offset:start_date:end_date:era_name:era_format</i>
4991		according to the definitions below. There can be as many era description segments as are necessary to describe the different eras. Era description segments are separated by <semicolon> characters.
4992		
4993		
4994	<i>direction</i>	Either a '+' or a '-' character. The '+' character shall indicate that years closer to the <i>start_date</i> have lower numbers than those closer to the <i>end_date</i> . The '-' character shall indicate that years closer to the <i>start_date</i> have higher numbers than those closer to the <i>end_date</i> .
4995		
4996		
4997		
4998		
4999	<i>offset</i>	The number of the year closest to the <i>start_date</i> in the era.
5000	<i>start_date</i>	A date in the form <i>yyyy/mm/dd</i> , where <i>yyyy</i> , <i>mm</i> , and <i>dd</i> are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 shall be represented as negative numbers.
5001		
5002		
5003	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> , or one of the two special values "-*" or "+*". The value "-*" shall indicate that the ending date is the beginning of time. The value "+*" shall indicate that the ending date is the end of time.
5004		
5005		
5006		
5007	<i>era_name</i>	The era, corresponding to the %EC conversion specification.
5008	<i>era_format</i>	The format of the year in the era, corresponding to the %EY conversion specification.
5009		
5010	ERA_D_FMT	The era date format.
5011	ERA_T_FMT	The locale's appropriate alternative time format, corresponding to the %EX conversion specification.
5012		
5013	ERA_D_T_FMT	The locale's appropriate alternative date and time format, corresponding to the %Ec conversion specification.
5014		

5015 ALT_DIGITS The alternative symbols for digits, corresponding to the %O conversion
 5016 specification modifier. The value consists of <semicolon>-separated symbols.
 5017 The first is the alternative symbol corresponding to zero, the second is the
 5018 symbol corresponding to one, and so on. Up to 100 alternative symbols may
 5019 be specified.

5020 7.3.5.3 LC_TIME Category in the POSIX Locale

5021 The LC_TIME category definition of the POSIX locale follows; the code listing depicts the
 5022 *localedef* input; the table represents the same information with the addition of *localedef* keywords,
 5023 conversion specifiers used by the *date* utility and the *strftime()*, *wcsftime()*, and *strptime()*
 5024 functions, and *nl_langinfo()* constants.

```

5025 LC_TIME
5026 # This is the POSIX locale definition for
5027 # the LC_TIME category.
5028 #
5029 # Abbreviated weekday names (%a)
5030 abday      "<S><u><n>"; "<M><o><n>"; "<T><u><e>"; "<W><e><d>"; \
5031            "<T><h><u>"; "<F><r><i>"; "<S><a><t>"; \
5032 #
5033 # Full weekday names (%A)
5034 day        "<S><u><n><d><a><y>"; "<M><o><n><d><a><y>"; \
5035            "<T><u><e><s><d><a><y>"; "<W><e><d><n><e><s><d><a><y>"; \
5036            "<T><h><u><r><s><d><a><y>"; "<F><r><i><d><a><y>"; \
5037            "<S><a><t><u><r><d><a><y>"; \
5038 #
5039 # Abbreviated month names (%b)
5040 abmon      "<J><a><n>"; "<F><e><b>"; "<M><a><r>"; \
5041            "<A><p><r>"; "<M><a><y>"; "<J><u><n>"; \
5042            "<J><u><l>"; "<A><u><g>"; "<S><e><p>"; \
5043            "<O><c><t>"; "<N><o><v>"; "<D><e><c>"; \
5044 #
5045 # Full month names (%B)
5046 mon        "<J><a><n><u><a><r><y>"; "<F><e><b><r><u><a><r><y>"; \
5047            "<M><a><r><c><h>"; "<A><p><r><i><l>"; \
5048            "<M><a><y>"; "<J><u><n><e>"; \
5049            "<J><u><l><y>"; "<A><u><g><u><s><t>"; \
5050            "<S><e><p><t><e><m><b><e><r>"; "<O><c><t><o><b><e><r>"; \
5051            "<N><o><v><e><m><b><e><r>"; "<D><e><c><e><m><b><e><r>"; \
5052 #
5053 # Equivalent of AM/PM (%p)      "AM"; "PM"
5054 am_pm      "<A><M>"; "<P><M>"
5055 #
5056 # Appropriate date and time representation (%c)
5057 #      "%a %b %e %H:%M:%S %Y"
5058 d_t_fmt    "<percent-sign><a><space><percent-sign><b>\
5059            <space><percent-sign><e><space><percent-sign><H>\
5060            <colon><percent-sign><M><colon><percent-sign><S>\
5061            <space><percent-sign><Y>"
5062 #
5063 # Appropriate date representation (%x)      "%m/%d/%Y"
5064 d_fmt      "<percent-sign><m><slash><percent-sign><d>\

```


Locale

Locale Definition

```

5065 <slash><percent-sign><y>"
5066 #
5067 # Appropriate time representation (%X) "%H:%M:%S"
5068 t_fmt "<percent-sign><H><colon><percent-sign><M>\
5069 <colon><percent-sign><S>"
5070 #
5071 # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
5072 t_fmt_ampm "<percent-sign><I><colon><percent-sign><M><colon>\
5073 <percent-sign><S><space><percent-sign><p>"
5074 #
5075 END LC_TIME

```

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
d_t_fmt	D_T_FMT	%c	"%a %b %e %H:%M:%S %Y"
d_fmt	D_FMT	%x	"%m/%d/%y"
t_fmt	T_FMT	%X	"%H:%M:%S"
am_pm	AM_STR	%p	"AM"
am_pm	PM_STR	%p	"PM"
t_fmt_ampm	T_FMT_AMP	%r	"%I:%M:%S %p"
day	DAY_1	%A	"Sunday"
day	DAY_2	%A	"Monday"
day	DAY_3	%A	"Tuesday"
day	DAY_4	%A	"Wednesday"
day	DAY_5	%A	"Thursday"
day	DAY_6	%A	"Friday"
day	DAY_7	%A	"Saturday"
abday	ABDAY_1	%a	"Sun"
abday	ABDAY_2	%a	"Mon"
abday	ABDAY_3	%a	"Tue"
abday	ABDAY_4	%a	"Wed"
abday	ABDAY_5	%a	"Thu"
abday	ABDAY_6	%a	"Fri"
abday	ABDAY_7	%a	"Sat"
mon	MON_1	%B	"January"
mon	MON_2	%B	"February"
mon	MON_3	%B	"March"
mon	MON_4	%B	"April"
mon	MON_5	%B	"May"
mon	MON_6	%B	"June"
mon	MON_7	%B	"July"
mon	MON_8	%B	"August"
mon	MON_9	%B	"September"
mon	MON_10	%B	"October"
mon	MON_11	%B	"November"
mon	MON_12	%B	"December"
abmon	ABMON_1	%b	"Jan"
abmon	ABMON_2	%b	"Feb"
abmon	ABMON_3	%b	"Mar"
abmon	ABMON_4	%b	"Apr"
abmon	ABMON_5	%b	"May"
abmon	ABMON_6	%b	"Jun"

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
abmon	ABMON_7	%b	"Jul "
abmon	ABMON_8	%b	"Aug "
abmon	ABMON_9	%b	"Sep "
abmon	ABMON_10	%b	"Oct "
abmon	ABMON_11	%b	"Nov "
abmon	ABMON_12	%b	"Dec "
era	ERA	%EC, %EY, %EY	N/A
era_d_fmt	ERA_D_FMT	%Ex	N/A
era_t_fmt	ERA_T_FMT	%EX	N/A
era_d_t_fmt	ERA_D_T_FMT	%Ec	N/A
alt_digits	ALT_DIGITS	%O	N/A

The entry N/A indicates the value is not available in the POSIX locale.

7.3.6 LC_MESSAGES

The *LC_MESSAGES* category shall define the format and values used by various utilities for affirmative and negative responses. This information is available through the *nl_langinfo()* function.

The message catalog used by the standard utilities and selected by the *catopen()* function shall be determined by the setting of *NLSPATH*; see Chapter 8 (on page 173). The *LC_MESSAGES* category can be specified as part of an *NLSPATH* substitution field.

The following keywords shall be recognized as part of the locale definition file.

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

Note: This is a *localedef* keyword, unavailable through *nl_langinfo()*.

yesexpr The operand consists of an extended regular expression (see Section 9.4, on page 188) that describes the acceptable affirmative response to a question expecting an affirmative or negative response.

noexpr The operand consists of an extended regular expression that describes the acceptable negative response to a question expecting an affirmative or negative response.

7.3.6.1 LC_MESSAGES Category in the POSIX Locale

The format and values for affirmative and negative responses of the POSIX locale follow; the code listing depicting the *localedef* input, the table representing the same information with the addition of *nl_langinfo()* constants.

```
LC_MESSAGES
# This is the POSIX locale definition for
# the LC_MESSAGES category.
#
yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
#
noexpr "<circumflex><left-square-bracket><n><N><right-square-bracket>"
#
END LC_MESSAGES
```


	localedef Keyword	langinfo Constant	POSIX Locale Value
5160			
5161	yesexpr	YESEXPR	"^[yY]"
5162	noexpr	NOEXPR	"^[nN]"

7.4 Locale Definition Grammar

The grammar and lexical conventions in this section shall together describe the syntax for the locale definition source. The general conventions for this style of grammar are described in XCU Section 1.3 (on page 2287). The grammar shall take precedence over the text in this chapter.

7.4.1 Locale Lexical Conventions

The lexical conventions for the locale definition grammar are described in this section.

The following tokens shall be processed (in addition to those string constants shown in the grammar):

LOC_NAME	A string of characters representing the name of a locale.
CHAR	Any single character.
NUMBER	A decimal number, represented by one or more decimal digits.
COLLSYMBOL	A symbolic name, enclosed between angle brackets. The string cannot duplicate any charmap symbol defined in the current charmap (if any), or a COLLELEMENT symbol.
COLLELEMENT	A symbolic name, enclosed between angle brackets, which cannot duplicate either any charmap symbol or a COLLSYMBOL symbol.
CHARCLASS	A string of alphanumeric characters from the portable character set, the first of which is not a digit, consisting of at least one and at most {CHARCLASS_NAME_MAX} bytes, and optionally surrounded by double-quotes.
CHARSYMBOL	A symbolic name, enclosed between angle brackets, from the current charmap (if any).
OCTAL_CHAR	One or more octal representations of the encoding of each byte in a single character. The octal representation consists of an escape character (normally a <backslash>) followed by two or more octal digits.
HEX_CHAR	One or more hexadecimal representations of the encoding of each byte in a single character. The hexadecimal representation consists of an escape character followed by the constant <i>x</i> and two or more hexadecimal digits.
DECIMAL_CHAR	One or more decimal representations of the encoding of each byte in a single character. The decimal representation consists of an escape character followed by a character 'd' and two or more decimal digits.

5197	ELLIPSIS	The string " . . . ".
5198	EXTENDED_REG_EXP	An extended regular expression as defined in the grammar in Section 9.5 (on page 191).
5199		
5200	EOL	The line termination character <newline>.

5201 7.4.2 Locale Grammar

5202 This section presents the grammar for the locale definition.

5203	%token	LOC_NAME
5204	%token	CHAR
5205	%token	NUMBER
5206	%token	COLLSYMBOL COLLELEMENT
5207	%token	CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
5208	%token	ELLIPSIS
5209	%token	EXTENDED_REG_EXP
5210	%token	EOL
5211	%start	locale_definition
5212	%%	
5213	locale_definition	: global_statements locale_categories
5214		locale_categories
5215		;
5216	global_statements	: global_statements symbol_redefine
5217		symbol_redefine
5218		;
5219	symbol_redefine	: 'escape_char' CHAR EOL
5220		'comment_char' CHAR EOL
5221		;
5222	locale_categories	: locale_categories locale_category
5223		locale_category
5224		;
5225	locale_category	: lc_ctype lc_collate lc_messages
5226		lc_monetary lc_numeric lc_time
5227		;
5228	/* The following grammar rules are common to all categories */	
5229	char_list	: char_list char_symbol
5230		char_symbol
5231		;
5232	char_symbol	: CHAR CHARSYMBOL
5233		OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
5234		;
5235	elem_list	: elem_list char_symbol
5236		elem_list COLLSYMBOL
5237		elem_list COLLELEMENT
5238		char_symbol
5239		COLLSYMBOL

*Locale**Locale Definition Grammar*

```

5240             | COLLELEMENT
5241             ;
5242     symb_list      : symb_list COLLSYMBOL
5243                     | COLLSYMBOL
5244                     ;
5245     locale_name     : LOC_NAME
5246                     | '"" LOC_NAME '""
5247                     ;
5248     /* The following is the LC_CTYPE category grammar */
5249     lc_ctype        : ctype_hdr ctype_keywords      ctype_tlr
5250                     | ctype_hdr 'copy' locale_name EOL ctype_tlr
5251                     ;
5252     ctype_hdr       : 'LC_CTYPE' EOL
5253                     ;
5254     ctype_keywords  : ctype_keywords ctype_keyword
5255                     | ctype_keyword
5256                     ;
5257     ctype_keyword   : charclass_keyword charclass_list EOL
5258                     | charconv_keyword charconv_list EOL
5259                     | 'charclass' charclass_namelist EOL
5260                     ;
5261     charclass_namelist : charclass_namelist ';' CHARCLASS
5262                       | CHARCLASS
5263                       ;
5264     charclass_keyword : 'upper' | 'lower' | 'alpha' | 'digit'
5265                       | 'punct' | 'xdigit' | 'space' | 'print'
5266                       | 'graph' | 'blank' | 'cntrl' | 'alnum'
5267                       | CHARCLASS
5268                       ;
5269     charclass_list   : charclass_list ';' char_symbol
5270                       | charclass_list ';' ELLIPSIS ';' char_symbol
5271                       | char_symbol
5272                       ;
5273     charconv_keyword : 'toupper'
5274                       | 'tolower'
5275                       ;
5276     charconv_list    : charconv_list ';' charconv_entry
5277                       | charconv_entry
5278                       ;
5279     charconv_entry   : '(' char_symbol ',' char_symbol ')'
5280                       ;
5281     ctype_tlr        : 'END' 'LC_CTYPE' EOL
5282                       ;
5283     /* The following is the LC_COLLATE category grammar */
5284     lc_collate       : collate_hdr collate_keywords      collate_tlr

```

Locale Definition Grammar

Locale

```

5285         | collate_hdr 'copy' locale_name EOL collate_tlr
5286         ;
5287     collate_hdr      : 'LC_COLLATE' EOL
5288         ;
5289     collate_keywords  :          order_statements
5290         | opt_statements order_statements
5291         ;
5292     opt_statements    : opt_statements collating_symbols
5293         | opt_statements collating_elements
5294         | collating_symbols
5295         | collating_elements
5296         ;
5297     collating_symbols : 'collating-symbol' COLLSYMBOL EOL
5298         ;
5299     collating_elements : 'collating-element' COLLELEMENT
5300         | 'from' ' ' elem_list ' ' EOL
5301         ;
5302     order_statements  : order_start collation_order order_end
5303         ;
5304     order_start       : 'order_start' EOL
5305         | 'order_start' order_opts EOL
5306         ;
5307     order_opts        : order_opts ';' order_opt
5308         | order_opt
5309         ;
5310     order_opt         : order_opt ',' opt_word
5311         | opt_word
5312         ;
5313     opt_word          : 'forward' | 'backward' | 'position'
5314         ;
5315     collation_order    : collation_order collation_entry
5316         | collation_entry
5317         ;
5318     collation_entry    : COLLSYMBOL EOL
5319         | collation_element weight_list EOL
5320         | collation_element          EOL
5321         ;
5322     collation_element  : char_symbol
5323         | COLLELEMENT
5324         | ELLIPSIS
5325         | 'UNDEFINED'
5326         ;
5327     weight_list       : weight_list ';' weight_symbol
5328         | weight_list ';'
5329         | weight_symbol
5330         ;

```

*Locale**Locale Definition Grammar*

```

5331     weight_symbol      : /* empty */
5332                          | char_symbol
5333                          | COLLSYMBOL
5334                          | ''' elem_list '''
5335                          | ''' symb_list '''
5336                          | ELLIPSIS
5337                          | 'IGNORE'
5338                          ;
5339     order_end            : 'order_end' EOL
5340                          ;
5341     collate_tlr          : 'END' 'LC_COLLATE' EOL
5342                          ;
5343     /* The following is the LC_MESSAGES category grammar */
5344     lc_messages           : messages_hdr messages_keywords messages_tlr
5345                          | messages_hdr 'copy' locale_name EOL messages_tlr
5346                          ;
5347     messages_hdr         : 'LC_MESSAGES' EOL
5348                          ;
5349     messages_keywords     : messages_keywords messages_keyword
5350                          | messages_keyword
5351                          ;
5352     messages_keyword      : 'yesexpr' ''' EXTENDED_REG_EXP ''' EOL
5353                          | 'noexpr' ''' EXTENDED_REG_EXP ''' EOL
5354                          ;
5355     messages_tlr          : 'END' 'LC_MESSAGES' EOL
5356                          ;
5357     /* The following is the LC_MONETARY category grammar */
5358     lc_monetary           : monetary_hdr monetary_keywords monetary_tlr
5359                          | monetary_hdr 'copy' locale_name EOL monetary_tlr
5360                          ;
5361     monetary_hdr          : 'LC_MONETARY' EOL
5362                          ;
5363     monetary_keywords     : monetary_keywords monetary_keyword
5364                          | monetary_keyword
5365                          ;
5366     monetary_keyword      : mon_keyword_string mon_string EOL
5367                          | mon_keyword_char NUMBER EOL
5368                          | mon_keyword_char '-1' EOL
5369                          | mon_keyword_grouping mon_group_list EOL
5370                          ;
5371     mon_keyword_string    : 'int_curr_symbol' | 'currency_symbol'
5372                          | 'mon_decimal_point' | 'mon_thousands_sep'
5373                          | 'positive_sign' | 'negative_sign'
5374                          ;
5375     mon_string            : ''' char_list '''

```

Locale Definition Grammar

Locale

```

5376         | '""'
5377         ;
5378     mon_keyword_char    : 'int_frac_digits' | 'frac_digits'
5379         | 'p_cs_precedes' | 'p_sep_by_space'
5380         | 'n_cs_precedes' | 'n_sep_by_space'
5381         | 'p_sign_posn' | 'n_sign_posn'
5382         | 'int_p_cs_precedes' | 'int_p_sep_by_space'
5383         | 'int_n_cs_precedes' | 'int_n_sep_by_space'
5384         | 'int_p_sign_posn' | 'int_n_sign_posn'
5385         ;
5386     mon_keyword_grouping : 'mon_grouping'
5387         ;
5388     mon_group_list      : NUMBER
5389         | mon_group_list ';' NUMBER
5390         ;
5391     monetary_tlr        : 'END' 'LC_MONETARY' EOL
5392         ;
5393     /* The following is the LC_NUMERIC category grammar */
5394     lc_numeric           : numeric_hdr numeric_keywords      numeric_tlr
5395         | numeric_hdr 'copy' locale_name EOL numeric_tlr
5396         ;
5397     numeric_hdr          : 'LC_NUMERIC' EOL
5398         ;
5399     numeric_keywords     : numeric_keywords numeric_keyword
5400         | numeric_keyword
5401         ;
5402     numeric_keyword      : num_keyword_string num_string EOL
5403         | num_keyword_grouping num_group_list EOL
5404         ;
5405     num_keyword_string   : 'decimal_point'
5406         | 'thousands_sep'
5407         ;
5408     num_string           : '""' char_list '""'
5409         | '""'
5410         ;
5411     num_keyword_grouping : 'grouping'
5412         ;
5413     num_group_list       : NUMBER
5414         | num_group_list ';' NUMBER
5415         ;
5416     numeric_tlr          : 'END' 'LC_NUMERIC' EOL
5417         ;
5418     /* The following is the LC_TIME category grammar */
5419     lc_time              : time_hdr time_keywords      time_tlr
5420         | time_hdr 'copy' locale_name EOL time_tlr

```

*Locale**Locale Definition Grammar*

```

5421                                     ;
5422     time_hdr                        : 'LC_TIME' EOL
5423                                     ;
5424     time_keywords                    : time_keywords time_keyword
5425                                     | time_keyword
5426                                     ;
5427     time_keyword                     : time_keyword_name time_list EOL
5428                                     | time_keyword_fmt time_string EOL
5429                                     | time_keyword_opt time_list EOL
5430                                     ;
5431     time_keyword_name                : 'abday' | 'day' | 'abmon' | 'mon'
5432                                     ;
5433     time_keyword_fmt                 : 'd_t_fmt' | 'd_fmt' | 't_fmt'
5434                                     | 'am_pm' | 't_fmt_ampm'
5435                                     ;
5436     time_keyword_opt                 : 'era' | 'era_d_fmt' | 'era_t_fmt'
5437                                     | 'era_d_t_fmt' | 'alt_digits'
5438                                     ;
5439     time_list                        : time_list ';' time_string
5440                                     | time_string
5441                                     ;
5442     time_string                      : '"' char_list '"'
5443                                     ;
5444     time_tlr                         : 'END' 'LC_TIME' EOL
5445                                     ;

```

(blank page)

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

Environment Variables

8.1 Environment Variable Definition

Environment variables defined in this chapter affect the operation of multiple utilities, functions, and applications. There are other environment variables that are of interest only to specific utilities. Environment variables that apply to a single utility only are defined as part of the utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in the Shell and Utilities volume of POSIX.1-2008 for information on environment variable usage.

The value of an environment variable is a string of characters. For a C-language program, an array of strings called the environment shall be made available when a process begins. The array is pointed to by the external variable *environ*, which is defined as:

```
extern char **environ;
```

These strings have the form *name=value*; *names* shall not contain the character '='. For values to be portable across systems conforming to POSIX.1-2008, the value shall be composed of characters from the portable character set (except NUL and as indicated below). There is no meaning associated with the order of strings in the environment. If more than one string in an environment of a process has the same *name*, the consequences are undefined.

Environment variable names used by the utilities in the Shell and Utilities volume of POSIX.1-2008 consist solely of uppercase letters, digits, and the <underscore> ('_') from the characters defined in Table 6-1 (on page 125) and do not begin with a digit. Other characters may be permitted by an implementation; applications shall tolerate the presence of such names. Uppercase and lowercase letters shall retain their unique identities and shall not be folded together. The name space of environment variable names containing lowercase letters is reserved for applications. Applications can define any environment variables with names from this name space without modifying the behavior of the standard utilities.

Note: Other applications may have difficulty dealing with environment variable names that start with a digit. For this reason, use of such names is not recommended anywhere.

The *values* that the environment variables may be assigned are not restricted except that they are considered to end with a null byte and the total space used to store the environment and the arguments to the process is limited to {ARG_MAX} bytes.

Other *name=value* pairs may be placed in the environment by, for example, calling any of the *setenv()*, *unsetenv()*, or *putenv()* functions, manipulating the *environ* variable, or by using *envp* arguments when creating a process; see *exec* in the System Interfaces volume of POSIX.1-2008.

It is unwise to conflict with certain variables that are frequently exported by widely used command interpreters and applications:

*Environment Variable Definition**Environment Variables*

5481	ARFLAGS	IFS	MAILPATH	PS1
5482	CC	LANG	MAILRC	PS2
5483	CDPATH	LC_ALL	MAKEFLAGS	PS3
5484	CFLAGS	LC_COLLATE	MAKESHELL	PS4
5485	CHARSET	LC_CTYPE	MANPATH	PWD
5486	COLUMNS	LC_MESSAGES	MBOX	RANDOM
5487	DATMSK	LC_MONETARY	MORE	SECONDS
5488	DEAD	LC_NUMERIC	MSGVERB	SHELL
5489	EDITOR	LC_TIME	NLSPATH	TERM
5490	ENV	LDFLAGS	NPROC	TERMCAP
5491	EXINIT	LEX	OLDPWD	TERMINFO
5492	FC	LFLAGS	OPTARG	TMPDIR
5493	FCEDIT	LINENO	OPTERR	TZ
5494	FFLAGS	LINES	OPTIND	USER
5495	GET	LISTER	PAGER	VISUAL
5496	GFLAGS	LOGNAME	PATH	YACC
5497	HISTFILE	LPDEST	PPID	YFLAGS
5498	HISTORY	MAIL	PRINTER	
5499	HISTSIZE	MAILCHECK	PROCLANG	
5500	HOME	MAILER	PROJECTDIR	

If the variables in the following two sections are present in the environment during the execution of an application or utility, they shall be given the meaning described below. Some are placed into the environment by the implementation at the time the user logs in; all can be added or changed by the user or any ancestor of the current process. The implementation adds or changes environment variables named in POSIX.1-2008 only as specified in POSIX.1-2008. If they are defined in the application's environment, the utilities in the Shell and Utilities volume of POSIX.1-2008 and the functions in the System Interfaces volume of POSIX.1-2008 assume they have the specified meaning. Conforming applications shall not set these environment variables to have meanings other than as described. See *getenv()* (on page 1008) and XCU Section 2.12 (on page 2331) for methods of accessing these variables.

8.2 Internationalization Variables

This section describes environment variables that are relevant to the operation of internationalized interfaces described in POSIX.1-2008.

Users may use the following environment variables to announce specific localization requirements to applications. Applications can retrieve this information using the *setlocale()* function to initialize the correct behavior of the internationalized interfaces. The descriptions of the internationalization environment variables describe the resulting behavior only when the application locale is initialized in this way. The use of the internationalization variables by utilities described in the Shell and Utilities volume of POSIX.1-2008 is described in the ENVIRONMENT VARIABLES section for those utilities in addition to the global effects described in this section.

LANG This variable shall determine the locale category for native language, local customs, and coded character set in the absence of the *LC_ALL* and other *LC_** (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) environment variables. This can be used by applications to determine the language to use for error messages and instructions, collating sequences, date formats, and so on.

Environment Variables

Internationalization Variables

5528	<i>LC_ALL</i>	This variable shall determine the values for all locale categories. The value of the <i>LC_ALL</i> environment variable has precedence over any of the other environment variables starting with <i>LC_</i> (<i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> , <i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i>) and the <i>LANG</i> environment variable.
5529		
5530		
5531		
5532		
5533	<i>LC_COLLATE</i>	This variable shall determine the locale category for character collation. It determines collation information for regular expressions and sorting, including equivalence classes and multi-character collating elements, in various utilities and the <i>strcoll()</i> and <i>strxfrm()</i> functions. Additional semantics of this variable, if any, are implementation-defined.
5534		
5535		
5536		
5537		
5538	<i>LC_CTYPE</i>	This variable shall determine the locale category for character handling functions, such as <i>tolower()</i> , <i>toupper()</i> , and <i>isalpha()</i> . This environment variable determines the interpretation of sequences of bytes of text data as characters (for example, single as opposed to multi-byte characters), the classification of characters (for example, alpha, digit, graph), and the behavior of character classes. Additional semantics of this variable, if any, are implementation-defined.
5539		
5540		
5541		
5542		
5543		
5544		
5545	<i>LC_MESSAGES</i>	This variable shall determine the locale category for processing affirmative and negative responses and the language and cultural conventions in which messages should be written. It also affects the behavior of the <i>catopen()</i> function in determining the message catalog. Additional semantics of this variable, if any, are implementation-defined. The language and cultural conventions of diagnostic and informative messages whose format is unspecified by POSIX.1-2008 should be affected by the setting of <i>LC_MESSAGES</i> .
5546		
5547		
5548		
5549		
5550		
5551		
5552		
5553	<i>LC_MONETARY</i>	This variable shall determine the locale category for monetary-related numeric formatting information. Additional semantics of this variable, if any, are implementation-defined.
5554		
5555		
5556	<i>LC_NUMERIC</i>	This variable shall determine the locale category for numeric formatting (for example, thousands separator and radix character) information in various utilities as well as the formatted I/O operations in <i>printf()</i> and <i>scanf()</i> and the string conversion functions in <i>strtod()</i> . Additional semantics of this variable, if any, are implementation-defined.
5557		
5558		
5559		
5560		
5561	<i>LC_TIME</i>	This variable shall determine the locale category for date and time formatting information. It affects the behavior of the time functions in <i>strftime()</i> . Additional semantics of this variable, if any, are implementation-defined.
5562		
5563		
5564	<i>NLSPATH</i>	This variable shall contain a sequence of templates that the <i>catopen()</i> function uses when attempting to locate message catalogs. Each template consists of an optional prefix, one or more conversion specifications, a filename, and an optional suffix.
5565		
5566		
5567		
5568		For example:
5569		<code>NLSPATH="/system/nlslib/%N.cat"</code>
5570		defines that <i>catopen()</i> should look for all message catalogs in the directory <code>/system/nlslib</code> , where the catalog name should be constructed from the <i>name</i> parameter passed to <i>catopen()</i> (<i>%N</i>), with the suffix <code>.cat</code> .
5571		
5572		
5573		Conversion specifications consist of a <code>' % '</code> symbol, followed by a single-letter keyword. The following keywords are currently defined:
5574		

5575	%N	The value of the <i>name</i> parameter passed to <i>catopen</i> ().
5576	%L	The value of the <i>LC_MESSAGES</i> category.
5577	%l	The <i>language</i> element from the <i>LC_MESSAGES</i> category.
5578	%t	The <i>territory</i> element from the <i>LC_MESSAGES</i> category.
5579	%c	The <i>codeset</i> element from the <i>LC_MESSAGES</i> category.
5580	%%	A single ' % ' character.
5581		An empty string is substituted if the specified value is not currently defined.
5582		The separators <underscore> (' _ ') and <period> (' . ') are not included in
5583		the %t and %c conversion specifications.
5584		Templates defined in <i>NLSPATH</i> are separated by <colon> characters (' : '). A
5585		leading or two adjacent <colon> characters (" : : ") is equivalent to specifying
5586		%N. For example:
5587		<code>NLSPATH=" : %N.cat : /nlslib/%L/%N.cat "</code>
5588		indicates to <i>catopen</i> () that it should look for the requested message catalog in
5589		<i>name</i> , <i>name.cat</i> , and <i>/nlslib/category/name.cat</i> , where <i>category</i> is the value of the
5590		<i>LC_MESSAGES</i> category of the current locale.
5591		Users should not set the <i>NLSPATH</i> variable unless they have a specific reason
5592		to override the default system path. Setting <i>NLSPATH</i> to override the default
5593		system path produces undefined results in the standard utilities and in
5594		applications with appropriate privileges.
5595		The environment variables <i>LANG</i> , <i>LC_ALL</i> , <i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,
5596		<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i> , and <i>NLSPATH</i> provide for the support of
5597		internationalized applications. The standard utilities shall make use of these environment
5598		variables as described in this section and the individual ENVIRONMENT VARIABLES sections
5599		for the utilities. If these variables specify locale categories that are not based upon the same
5600		underlying codeset, the results are unspecified.
5601		The values of locale categories shall be determined by a precedence order; the first condition met
5602		below determines the value:
5603	1.	If the <i>LC_ALL</i> environment variable is defined and is not null, the value of <i>LC_ALL</i> shall
5604		be used.
5605	2.	If the <i>LC_*</i> environment variable (<i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,
5606		<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i>) is defined and is not null, the value of the
5607		environment variable shall be used to initialize the category that corresponds to the
5608		environment variable.
5609	3.	If the <i>LANG</i> environment variable is defined and is not null, the value of the <i>LANG</i>
5610		environment variable shall be used.
5611	4.	If the <i>LANG</i> environment variable is not set or is set to the empty string, the
5612		implementation-defined default locale shall be used.
5613		If the locale value is "C" or "POSIX", the POSIX locale shall be used and the standard utilities
5614		behave in accordance with the rules in Section 7.2 (on page 136) for the associated category.
5615		If the locale value begins with a <slash>, it shall be interpreted as the pathname of a file that was
5616		created in the output format used by the <i>localedef</i> utility; see OUTPUT FILES under <i>localedef</i> .
5617		Referencing such a pathname shall result in that locale being used for the indicated category.

Environment Variables

Internationalization Variables

5618 XSI If the locale value has the form:

5619 `language[_territory] [.codeset]`

5620 it refers to an implementation-provided locale, where settings of language, territory, and codeset

5621 are implementation-defined.

5622 `LC_COLLATE`, `LC_CTYPE`, `LC_MESSAGES`, `LC_MONETARY`, `LC_NUMERIC`, and `LC_TIME` are

5623 defined to accept an additional field `@modifier`, which allows the user to select a specific instance

5624 of localization data within a single category (for example, for selecting the dictionary as opposed

5625 to the character ordering of data). The syntax for these environment variables is thus defined as:

5626 `[language[_territory] [.codeset] [@modifier]]`

5627 For example, if a user wanted to interact with the system in French, but required to sort German

5628 text files, `LANG` and `LC_COLLATE` could be defined as:

5629 `LANG=Fr_FR`

5630 `LC_COLLATE=De_DE`

5631 This could be extended to select dictionary collation (say) by use of the `@modifier` field; for

5632 example:

5633 `LC_COLLATE=De_DE@dict`

5634 An implementation may support other formats.

5635 If the locale value is not recognized by the implementation, the behavior is unspecified.

5636 At runtime, these values are bound to the locale of a process by calling the `setlocale()` function.

5637 Additional criteria for determining a valid locale name are implementation-defined.

8.3 Other Environment Variables

5638 **COLUMNS** This variable shall represent a decimal integer >0 used to indicate the user's

5639 preferred width in column positions for the terminal screen or window; see

5640 [Section 3.103](#) (on page 50). If this variable is unset or null, the implementation

5641 determines the number of columns, appropriate for the terminal or window,

5642 in an unspecified manner. When `COLUMNS` is set, any terminal-width

5643 information implied by `TERM` is overridden. Users and conforming

5644 applications should not set `COLUMNS` unless they wish to override the

5645 system selection and produce output unrelated to the terminal characteristics.

5646

5647 Users should not need to set this variable in the environment unless there is a

5648 specific reason to override the implementation's default behavior, such as to

5649 display data in an area arbitrarily smaller than the terminal or window.

5650 XSI **DATEMSK** Indicates the pathname of the template file used by `getdate()`.

5651 **HOME** The system shall initialize this variable at the time of login to be a pathname of

5652 the user's home directory. See [<pwd.h>](#).

5653 **LINES** This variable shall represent a decimal integer >0 used to indicate the user's

5654 preferred number of lines on a page or the vertical screen or window size in

5655 lines. A line in this case is a vertical measure large enough to hold the tallest

5656 character in the character set being displayed. If this variable is unset or null,

5657 the implementation determines the number of lines, appropriate for the

5658		terminal or window (size, terminal baud rate, and so on), in an unspecified manner. When <i>LINES</i> is set, any terminal-height information implied by <i>TERM</i> is overridden. Users and conforming applications should not set <i>LINES</i> unless they wish to override the system selection and produce output unrelated to the terminal characteristics.
5659		
5660		
5661		
5662		
5663		Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.
5664		
5665		
5666	<i>LOGNAME</i>	The system shall initialize this variable at the time of login to be the user's login name. See <pwd.h> . For a value of <i>LOGNAME</i> to be portable across implementations of POSIX.1-2008, the value should be composed of characters from the portable filename character set.
5667		
5668		
5669		
5670	XSI	<i>MSGVERB</i> Describes which message components shall be used in writing messages by <i>fmtmsg()</i> .
5671		
5672	<i>PATH</i>	This variable shall represent the sequence of path prefixes that certain functions and utilities apply in searching for an executable file known only by a filename. The prefixes shall be separated by a <colon> (' : '). When a non-zero-length prefix is applied to this filename, a <slash> shall be inserted between the prefix and the filename. A zero-length prefix is a legacy feature that indicates the current working directory. It appears as two adjacent <colon> characters (" : : "), as an initial <colon> preceding the rest of the list, or as a trailing <colon> following the rest of the list. A strictly conforming application shall use an actual pathname (such as .) to represent the current working directory in <i>PATH</i> . The list shall be searched from beginning to end, applying the filename to each prefix, until an executable file with the specified name and appropriate execution permissions is found. If the pathname being sought contains a <slash>, the search through the path prefixes shall not be performed. If the pathname begins with a <slash>, the specified path is resolved (see Section 4.12 , on page 111). If <i>PATH</i> is unset or is set to null, the path search is implementation-defined.
5673		
5674		
5675		
5676		
5677		
5678		
5679		
5680		
5681		
5682		
5683		
5684		
5685		
5686		
5687		
5688	<i>PWD</i>	This variable shall represent an absolute pathname of the current working directory. It shall not contain any components that are dot or dot-dot. The value is set by the <i>cd</i> utility, and by the <i>sh</i> utility during initialization.
5689		
5690		
5691	<i>SHELL</i>	This variable shall represent a pathname of the user's preferred command language interpreter. If this interpreter does not conform to the Shell Command Language in XCU Chapter 2 (on page 2297), utilities may behave differently from those described in POSIX.1-2008.
5692		
5693		
5694		
5695	<i>TMPDIR</i>	This variable shall represent a pathname of a directory made available for programs that need a place to create temporary files.
5696		
5697	<i>TERM</i>	This variable shall represent the terminal type for which output is to be prepared. This information is used by utilities and application programs wishing to exploit special capabilities specific to a terminal. The format and allowable values of this environment variable are unspecified.
5698		
5699		
5700		
5701	<i>TZ</i>	This variable shall represent timezone information. The contents of the environment variable named <i>TZ</i> shall be used by the <i>ctime()</i> , <i>ctime_r()</i> , <i>localtime()</i> , <i>localtime_r()</i> , <i>strftime()</i> , <i>mktime()</i> , functions, and by various utilities, to override the default timezone. The value of <i>TZ</i> has one of the two forms (spaces inserted for clarity):
5702		
5703		
5704		
5705		

Environment Variables

Other Environment Variables

5706 : *characters*

5707 or:

5708 *std offset dst offset, rule*

5709 If *TZ* is of the first format (that is, if the first character is a <colon>), the
5710 characters following the <colon> are handled in an implementation-defined
5711 manner.

5712 The expanded format (for all *TZ*s whose value does not have a <colon> as the
5713 first character) is as follows:

5714 *stdoffset* [*dst* [*offset*] [, *start* [/time], *end* [/time]]]

5715 Where:

5716 *std* and *dst* Indicate no less than three, nor more than {TZNAME_MAX},
5717 bytes that are the designation for the standard (*std*) or the
5718 alternative (*dst*—such as Daylight Savings Time) timezone. Only
5719 *std* is required; if *dst* is missing, then the alternative time does
5720 not apply in this locale.

5721 Each of these fields may occur in either of two formats quoted or
5722 unquoted:

5723 — In the quoted form, the first character shall be the <less-
5724 than-sign> ('<') character and the last character shall be
5725 the <greater-than-sign> ('>') character. All characters
5726 between these quoting characters shall be alphanumeric
5727 characters from the portable character set in the current
5728 locale, the <plus-sign> ('+') character, or the minus-sign
5729 ('-') character. The *std* and *dst* fields in this case shall not
5730 include the quoting characters.

5731 — In the unquoted form, all characters in these fields shall be
5732 alphabetic characters from the portable character set in the
5733 current locale.

5734 The interpretation of these fields is unspecified if either field is
5735 less than three bytes (except for the case when *dst* is missing),
5736 more than {TZNAME_MAX} bytes, or if they contain characters
5737 other than those specified.

5738 *offset* Indicates the value added to the local time to arrive at
5739 Coordinated Universal Time. The *offset* has the form:

5740 *hh* [: *mm*] [: *ss*]

5741 The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*)
5742 shall be required and may be a single digit. The *offset* following
5743 *std* shall be required. If no *offset* follows *dst*, the alternative time
5744 is assumed to be one hour ahead of standard time. One or more
5745 digits may be used; the value is always interpreted as a decimal
5746 number. The hour shall be between zero and 24, and the minutes
5747 (and seconds)—if present—between zero and 59. The result of
5748 using values outside of this range is unspecified. If preceded by
5749 a '-', the timezone shall be east of the Prime Meridian;
5750 otherwise, it shall be west (which may be indicated by an
5751 optional preceding '+').

5752	<i>rule</i>	Indicates when to change to and back from the alternative time.
5753		The <i>rule</i> has the form:
5754		<code>date[/time], date[/time]</code>
5755		where the first <i>date</i> describes when the change from standard to
5756		alternative time occurs and the second <i>date</i> describes when the
5757		change back happens. Each <i>time</i> field describes when, in current
5758		local time, the change to the other time is made.
5759		The format of <i>date</i> is one of the following:
5760	<i>Jn</i>	The Julian day <i>n</i> ($1 \leq n \leq 365$). Leap days shall not be
5761		counted. That is, in all years—including leap years—
5762		February 28 is day 59 and March 1 is day 60. It is
5763		impossible to refer explicitly to the occasional February
5764		29.
5765	<i>n</i>	The zero-based Julian day ($0 \leq n \leq 365$). Leap days shall
5766		be counted, and it is possible to refer to February 29.
5767	<i>Mm.n.d</i>	The <i>d</i> 'th day ($0 \leq d \leq 6$) of week <i>n</i> of month <i>m</i> of the
5768		year ($1 \leq n \leq 5$, $1 \leq m \leq 12$, where week 5 means “the last
5769		<i>d</i> day in month <i>m</i> ” which may occur in either the fourth
5770		or the fifth week). Week 1 is the first week in which the
5771		<i>d</i> 'th day occurs. Day zero is Sunday.
5772		The <i>time</i> has the same format as <i>offset</i> except that no leading sign
5773		('−' or '+') is allowed. The default, if <i>time</i> is not given, shall be
5774		02:00:00.

Regular Expressions

Regular Expressions (REs) provide a mechanism to select specific strings from a set of character strings.

Regular expressions are a context-independent syntax that can represent a wide variety of character sets and character set orderings, where these character sets are interpreted according to the current locale. While many regular expressions can be interpreted differently depending on the current locale, many features, such as character class expressions, provide for contextual invariance across locales.

The Basic Regular Expression (BRE) notation and construction rules in Section 9.3 (on page 183) shall apply to most utilities supporting regular expressions. Some utilities, instead, support the Extended Regular Expressions (ERE) described in Section 9.4 (on page 188); any exceptions for both cases are noted in the descriptions of the specific utilities using regular expressions. Both BREs and EREs are supported by the Regular Expression Matching interface in the System Interfaces volume of POSIX.1-2008 under *regcomp()*, *regex()*, and related functions.

9.1 Regular Expression Definitions

For the purposes of this section, the following definitions shall apply:

entire regular expression

The concatenated set of one or more BREs or EREs that make up the pattern specified for string selection.

matched

A sequence of zero or more characters shall be said to be matched by a BRE or ERE when the characters in the sequence correspond to a sequence of characters defined by the pattern.

Matching shall be based on the bit pattern used for encoding the character, not on the graphic representation of the character. This means that if a character set contains two or more encodings for a graphic symbol, or if the strings searched contain text encoded in more than one codeset, no attempt is made to search for any other representation of the encoded symbol. If that is required, the user can specify equivalence classes containing all variations of the desired graphic symbol.

The search for a matching sequence starts at the beginning of a string and stops when the first sequence matching the expression is found, where “first” is defined to mean “begins earliest in the string”. If the pattern permits a variable number of matching characters and thus there is more than one such sequence starting at that point, the longest such sequence is matched. For example, the BRE “bb*” matches the second to fourth characters of the string “abbbc”, and the ERE “(wee|week)(knights|night)” matches all ten characters of the string “weeknights”.

Consistent with the whole match being the longest of the leftmost matches, each subpattern, from left to right, shall match the longest possible string. For this purpose, a null string shall be considered to be longer than no match at all. For example, matching the BRE

5815 "\ (. * \) . *" against "abcdef", the subexpression " (\1) " is "abcdef", and matching
5816 the BRE "\ (a * \) *" against "bc", the subexpression " (\1) " is the null string.

5817 When a multi-character collating element in a bracket expression (see [Section 9.3.5](#), on page
5818 184) is involved, the longest sequence shall be measured in characters consumed from the
5819 string to be matched; that is, the collating element counts not as one element, but as the
5820 number of characters it matches.

5821 **BRE (ERE) matching a single character**

5822 A BRE or ERE that shall match either a single character or a single collating element.

5823 Only a BRE or ERE of this type that includes a bracket expression (see [Section 9.3.5](#), on page
5824 184) can match a collating element.

5825 **BRE (ERE) matching multiple characters**

5826 A BRE or ERE that shall match a concatenation of single characters or collating elements.

5827 Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE
5828 (ERE) special characters.

5829 **invalid**

5830 This section uses the term “invalid” for certain constructs or conditions. Invalid REs shall
5831 cause the utility or function using the RE to generate an error condition. When invalid is not
5832 used, violations of the specified syntax or semantics for REs produce undefined results: this
5833 may entail an error, enabling an extended syntax for that RE, or using the construct in error
5834 as literal characters to be matched. For example, the BRE construct "\ {1, 2, 3 \}" does not
5835 comply with the grammar. A conforming application cannot rely on it producing an error
5836 nor matching the literal characters "\ {1, 2, 3 \}".

5837 **9.2 Regular Expression General Requirements**

5838 The requirements in this section shall apply to both basic and extended regular expressions.

5839 The use of regular expressions is generally associated with text processing. REs (BREs and EREs)
5840 operate on text strings, that is, zero or more characters followed by an end-of-string delimiter
5841 (typically NUL). Some utilities employing regular expressions limit the processing to lines; that
5842 is, zero or more characters followed by a <newline>. In the regular expression processing
5843 described in POSIX.1-2008, the <newline> is regarded as an ordinary character and both a
5844 <period> and a non-matching list can match one. The Shell and Utilities volume of
5845 POSIX.1-2008 specifies within the individual descriptions of those standard utilities employing
5846 regular expressions whether they permit matching of <newline> characters; if not stated
5847 otherwise, the use of literal <newline> characters or any escape sequence equivalent produces
5848 undefined results. Those utilities (like *grep*) that do not allow <newline> characters to match are
5849 responsible for eliminating any <newline> from strings before matching against the RE. The
5850 *regcomp()* function in the System Interfaces volume of POSIX.1-2008, however, can provide
5851 support for such processing without violating the rules of this section.

5852 The interfaces specified in POSIX.1-2008 do not permit the inclusion of a NUL character in an RE
5853 or in the string to be matched. If during the operation of a standard utility a NUL is included in
5854 the text designated to be matched, that NUL may designate the end of the text string for the
5855 purposes of matching.

5856 When a standard utility or function that uses regular expressions specifies that pattern matching
5857 shall be performed without regard to the case (uppercase or lowercase) of either data or
5858 patterns, then when each character in the string is matched against the pattern, not only the

character, but also its case counterpart (if any), shall be matched. This definition of case-insensitive processing is intended to allow matching of multi-character collating elements as well as characters, as each character in the string is matched using both its cases. For example, in a locale where "Ch" is a multi-character collating element and where a matching list expression matches such elements, the RE "[[.Ch.]]" when matched against the string "char" is in reality matched against "ch", "Ch", "cH", and "CH".

The implementation shall support any regular expression that does not exceed 256 bytes in length.

9.3 Basic Regular Expressions

9.3.1 BREs Matching a Single Character or Collating Element

A BRE ordinary character, a special character preceded by a <backslash>, or a <period> shall match a single character. A bracket expression shall match a single character or a single collating element.

9.3.2 BRE Ordinary Characters

An ordinary character is a BRE that matches itself: any character in the supported character set, except for the BRE special characters listed in [Section 9.3.3](#).

The interpretation of an ordinary character preceded by a <backslash> ('\\') is undefined, except for:

- The characters ') ' , ' (' , ' { ' , and ' } '
- The digits 1 to 9 inclusive (see [Section 9.3.6](#), on page 186)
- A character inside a bracket expression

9.3.3 BRE Special Characters

A BRE special character has special properties in certain contexts. Outside those contexts, or when preceded by a <backslash>, such a character is a BRE that matches the special character itself. The BRE special characters and the contexts in which they have their special meaning are as follows:

. [\ The <period>, <left-square-bracket>, and <backslash> shall be special except when used in a bracket expression (see [Section 9.3.5](#), on page 184). An expression containing a ' [' that is not preceded by a <backslash> and is not part of a bracket expression produces undefined results.

* The <asterisk> shall be special except when used:

- In a bracket expression
- As the first character of an entire BRE (after an initial ' ^ ' , if any)

- 5892 — As the first character of a subexpression (after an initial '[^]', if any); see [Section](#)
 5893 [9.3.6](#) (on page 186)
- 5894 [^] The <circumflex> shall be special when used as:
- 5895 — An anchor (see [Section 9.3.8](#), on page 187)
- 5896 — The first character of a bracket expression (see [Section 9.3.5](#))
- 5897 \$ The <dollar-sign> shall be special when used as an anchor.

5898 9.3.4 Periods in BREs

5899 A <period> ('.'), when used outside a bracket expression, is a BRE that shall match any
 5900 character in the supported character set except NUL.

5901 9.3.5 RE Bracket Expression

5902 A bracket expression (an expression enclosed in square brackets "[]") is an RE that shall
 5903 match a single collating element contained in the non-empty set of collating elements
 5904 represented by the bracket expression.

5905 The following rules and definitions apply to bracket expressions:

- 5906 1. A bracket expression is either a matching list expression or a non-matching list
 5907 expression. It consists of one or more expressions: collating elements, collating symbols,
 5908 equivalence classes, character classes, or range expressions. The <right-square-bracket>
 5909 ('] ') shall lose its special meaning and represent itself in a bracket expression if it occurs
 5910 first in the list (after an initial <circumflex> ('[^]'), if any). Otherwise, it shall terminate
 5911 the bracket expression, unless it appears in a collating symbol (such as "[.] . ") or is the
 5912 ending <right-square-bracket> for a collating symbol, equivalence class, or character
 5913 class. The special characters '.', '*', '[', and '\\ ' (<period>, <asterisk>, <left-square-
 5914 bracket>, and <backslash>, respectively) shall lose their special meaning within a bracket
 5915 expression.
 5916 The character sequences "[. ", "[= ", and "[: " (<left-square-bracket> followed by a
 5917 <period>, <equals-sign>, or <colon>) shall be special inside a bracket expression and are
 5918 used to delimit collating symbols, equivalence class expressions, and character class
 5919 expressions. These symbols shall be followed by a valid expression and the matching
 5920 terminating sequence ".] ", "=] ", or ":] ", as described in the following items.
- 5921 2. A matching list expression specifies a list that shall match any single-character collating
 5922 element in any of the expressions represented in the list. The first character in the list shall
 5923 not be the <circumflex>; for example, "[abc]" is an RE that matches any of the
 5924 characters 'a', 'b', or 'c'. It is unspecified whether a matching list expression matches
 5925 a multi-character collating element that is matched by one of the expressions.
- 5926 3. A non-matching list expression begins with a <circumflex> ('[^]'), and specifies a list that
 5927 shall match any single-character collating element except for the expressions represented
 5928 in the list after the leading <circumflex>. For example, "[^abc]" is an RE that matches
 5929 any character except the characters 'a', 'b', or 'c'. It is unspecified whether a non-
 5930 matching list expression matches a multi-character collating element that is not matched
 5931 by any of the expressions. The <circumflex> shall have this special meaning only when it
 5932 occurs first in the list, immediately following the <left-square-bracket>.

Regular Expressions

Basic Regular Expressions

4. A collating symbol is a collating element enclosed within bracket-period ("[" and ".]") delimiters. Collating elements are defined as described in [Section 7.3.2.4](#) (on page 149). Conforming applications shall represent multi-character collating elements as collating symbols when it is necessary to distinguish them from a list of the individual characters that make up the multi-character collating element. For example, if the string "ch" is a collating element defined using the line:

```
collating-element <ch-digraph> from "<c><h>"
```

in the locale definition, the expression "[.ch.]" shall be treated as an RE containing the collating symbol 'ch', while "[ch]" shall be treated as an RE matching 'c' or 'h'. Collating symbols are recognized only inside bracket expressions. If the string is not a collating element in the current locale, the expression is invalid.

5. An equivalence class expression shall represent the set of collating elements belonging to an equivalence class, as described in [Section 7.3.2.4](#) (on page 149). Only primary equivalence classes shall be recognized. The class shall be expressed by enclosing any one of the collating elements in the equivalence class within bracket-equal ("[" and "=") delimiters. For example, if 'a', 'à', and '^' belong to the same equivalence class, then "[[=a=]b]", "[[=à=]b]", and "[[=^=]b]" are each equivalent to "[aà^b]". If the collating element does not belong to an equivalence class, the equivalence class expression shall be treated as a collating symbol.

6. A character class expression shall represent the union of two sets:

- a. The set of single-character collating elements whose characters belong to the character class, as defined in the `LC_CTYPE` category in the current locale.
- b. An unspecified set of multi-character collating elements.

All character classes specified in the current locale shall be recognized. A character class expression is expressed as a character class name enclosed within bracket-colon ("[:" and ":]") delimiters.

The following character class expressions shall be supported in all locales:

```
[ :alnum: ]    [ :cntrl: ]    [ :lower: ]    [ :space: ]
[ :alpha: ]    [ :digit: ]    [ :print: ]    [ :upper: ]
[ :blank: ]    [ :graph: ]    [ :punct: ]    [ :xdigit: ]
```

In addition, character class expressions of the form:

```
[ :name: ]
```

are recognized in those locales where the *name* keyword has been given a **charclass** definition in the `LC_CTYPE` category.

7. In the POSIX locale, a range expression represents the set of collating elements that fall between two elements in the collation sequence, inclusive. In other locales, a range expression has unspecified behavior: strictly conforming applications shall not rely on whether the range expression is valid, or on the set of collating elements matched. A range expression shall be expressed as the starting point and the ending point separated by a <hyphen> ('-').

In the following, all examples assume the POSIX locale.

The starting range point and the ending range point shall be a collating element or collating symbol. An equivalence class expression used as a starting or ending point of a range expression produces unspecified results. An equivalence class can be used portably within a bracket expression, but only outside the range. If the represented set of collating

elements is empty, it is unspecified whether the expression matches nothing, or is treated as invalid.

The interpretation of range expressions where the ending range point is also the starting range point of a subsequent range expression (for example, "[a-m-o]") is undefined.

The <hyphen> character shall be treated as itself if it occurs first (after an initial '^', if any) or last in the list, or as an ending range point in a range expression. As examples, the expressions "[ac]" and "[ac-]" are equivalent and match any of the characters 'a', 'c', or '-'; "[^ac]" and "[^ac-]" are equivalent and match any characters except 'a', 'c', or '-'; the expression "[%--]" matches any of the characters between '%' and '-' inclusive; the expression "[-@]" matches any of the characters between '-' and '@' inclusive; and the expression "[a--@]" is either invalid or equivalent to '@', because the letter 'a' follows the symbol '-' in the POSIX locale. To use a <hyphen> as the starting range point, it shall either come first in the bracket expression or be specified as a collating symbol; for example, "[[.-]-0]", which matches either a <right-square-bracket> or any character or collating element that collates between <hyphen> and 0, inclusive.

If a bracket expression specifies both '-' and ']', the ']' shall be placed first (after the '^', if any) and the '-' last within the bracket expression.

9.3.6 BREs Matching Multiple Characters

The following rules can be used to construct BREs matching multiple characters from BREs matching a single character:

1. The concatenation of BREs shall match the concatenation of the strings matched by each component of the BRE.
2. A subexpression can be defined within a BRE by enclosing it between the character pairs "\" and "\". Such a subexpression shall match whatever it would have matched without the "\" and "\" , except that anchoring within subexpressions is optional behavior; see [Section 9.3.8](#) (on page 187). Subexpressions can be arbitrarily nested.
3. The back-reference expression '\n' shall match the same (possibly empty) string of characters as was matched by a subexpression enclosed between "\" and "\" preceding the '\n'. The character 'n' shall be a digit from 1 through 9, specifying the *n*th subexpression (the one that begins with the *n*th "\" from the beginning of the pattern and ends with the corresponding paired "\"). The expression is invalid if less than *n* subexpressions precede the '\n'. The string matched by a contained subexpression shall be within the string matched by the containing subexpression. If the containing subexpression does not match, or if there is no match for the contained subexpression within the string matched by the containing subexpression, then back-reference expressions corresponding to the contained subexpression shall not match. When a subexpression matches more than one string, a back-reference expression corresponding to the subexpression shall refer to the last matched string. For example, the expression "^\\(.*)\\1\$" matches lines consisting of two adjacent appearances of the same string, and the expression "\\(a\\)*\\1" fails to match 'a', the expression "\\(a\\(b\\)*\\)*\\2" fails to match 'abab', and the expression "^\\(ab\\)*\\1\$" matches 'ababbabb', but fails to match 'ababbab'.
4. When a BRE matching a single character, a subexpression, or a back-reference is followed by the special character <asterisk> ('*'), together with that <asterisk> it shall match what zero or more consecutive occurrences of the BRE would match. For example, "[ab]*" and "[ab][ab]" are equivalent when matching the string "ab".

*Regular Expressions**Basic Regular Expressions*

5. When a BRE matching a single character, a subexpression, or a back-reference is followed by an interval expression of the format " $\{m\}$ ", " $\{m,\}$ ", or " $\{m,n\}$ ", together with that interval expression it shall match what repeated consecutive occurrences of the BRE would match. The values of m and n are decimal integers in the range $0 \leq m \leq n \leq \{RE\_DUP\_MAX\}$, where m specifies the exact or minimum number of occurrences and n specifies the maximum number of occurrences. The expression " $\{m\}$ " shall match exactly m occurrences of the preceding BRE, " $\{m,\}$ " shall match at least m occurrences, and " $\{m,n\}$ " shall match any number of occurrences between m and n , inclusive.

For example, in the string "abababcccccd" the BRE " $c\{3\}$ " is matched by characters seven to nine, the BRE " $\{ab\}\{4,\}$ " is not matched at all, and the BRE " $c\{1,3\}d$ " is matched by characters ten to thirteen.

The behavior of multiple adjacent duplication symbols ('*' and intervals) produces undefined results.

A subexpression repeated by an <asterisk> ('*') or an interval expression shall not match a null expression unless this is the only match for the repetition or it is necessary to satisfy the exact or minimum number of occurrences for the interval expression.

9.3.7 BRE Precedence

The order of precedence shall be as shown in the following table:

BRE Precedence (from high to low)	
Collation-related bracket symbols	[==] [::] [..]
Escaped characters	\<special character>
Bracket expression	[]
Subexpressions/back-references	\(\) \n
Single-character-BRE duplication	* \{m,n\}
Concatenation	
Anchoring	^ \$

9.3.8 BRE Expression Anchoring

A BRE can be limited to matching strings that begin or end a line; this is called "anchoring". The <circumflex> and <dollar-sign> special characters shall be considered BRE anchors in the following contexts:

1. A <circumflex> ('^') shall be an anchor when used as the first character of an entire BRE. The implementation may treat the <circumflex> as an anchor when used as the first character of a subexpression. The <circumflex> shall anchor the expression (or optionally subexpression) to the beginning of a string; only sequences starting at the first character of a string shall be matched by the BRE. For example, the BRE " ^ab " matches "ab" in the string "abcdef", but fails to match in the string "cdefab". The BRE " $\{^ab\}$ " may match the former string. A portable BRE shall escape a leading <circumflex> in a subexpression to match a literal circumflex.
2. A <dollar-sign> ('\$') shall be an anchor when used as the last character of an entire BRE. The implementation may treat a <dollar-sign> as an anchor when used as the last character of a subexpression. The <dollar-sign> shall anchor the expression (or optionally subexpression) to the end of the string being matched; the <dollar-sign> can be said to match the end-of-string following the last character.

3. A BRE anchored by both '`^`' and '`$`' shall match only an entire string. For example, the BRE "`^abcdef$`" matches strings consisting only of "`abcdef`".

9.4 Extended Regular Expressions

The extended regular expression (ERE) notation and construction rules shall apply to utilities defined as using extended regular expressions; any exceptions to the following rules are noted in the descriptions of the specific utilities using EREs.

9.4.1 EREs Matching a Single Character or Collating Element

An ERE ordinary character, a special character preceded by a `<backslash>` or a `<period>` shall match a single character. A bracket expression shall match a single character or a single collating element. An ERE matching a single character enclosed in parentheses shall match the same as the ERE without parentheses would have matched.

9.4.2 ERE Ordinary Characters

An ordinary character is an ERE that matches itself. An ordinary character is any character in the supported character set, except for the ERE special characters listed in [Section 9.4.3](#). The interpretation of an ordinary character preceded by a `<backslash>` (`'\'`) is undefined.

9.4.3 ERE Special Characters

An ERE special character has special properties in certain contexts. Outside those contexts, or when preceded by a `<backslash>`, such a character shall be an ERE that matches the special character itself. The extended regular expression special characters and the contexts in which they shall have their special meaning are as follows:

- `.` `[` `\` `(` The `<period>`, `<left-square-bracket>`, `<backslash>`, and `<left-parenthesis>` shall be special except when used in a bracket expression (see [Section 9.3.5](#), on page 184). Outside a bracket expression, a `<left-parenthesis>` immediately followed by a `<right-parenthesis>` produces undefined results.
- `)` The `<right-parenthesis>` shall be special when matched with a preceding `<left-parenthesis>`, both outside a bracket expression.
- `*` `+` `?` The `<asterisk>`, `<plus-sign>`, `<question-mark>`, and `<left-brace>` shall be special except when used in a bracket expression (see [Section 9.3.5](#), on page 184). Any of the following uses produce undefined results:
 - If these characters appear first in an ERE, or immediately following a `<vertical-line>`, `<circumflex>`, or `<left-parenthesis>`
 - If a `<left-brace>` is not part of a valid interval expression (see [Section 9.4.6](#), on page 189)
- `|` The `<vertical-line>` is special except when used in a bracket expression (see [Section 9.3.5](#), on page 184). A `<vertical-line>` appearing first or last in an ERE, or immediately following a `<vertical-line>` or a `<left-parenthesis>`, or immediately preceding a `<right-parenthesis>`, produces undefined results.

*Regular Expressions**Extended Regular Expressions*

- 6106 [^] The <circumflex> shall be special when used as:
- 6107 — An anchor (see [Section 9.4.9](#), on page 190)
- 6108 — The first character of a bracket expression (see [Section 9.3.5](#), on page 184)
- 6109 \$ The <dollar-sign> shall be special when used as an anchor.

6110 **9.4.4 Periods in EREs**

6111 A <period> ('.'), when used outside a bracket expression, is an ERE that shall match any
 6112 character in the supported character set except NUL.

6113 **9.4.5 ERE Bracket Expression**

6114 The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see [Section](#)
 6115 [9.3.5](#) (on page 184).

6116 **9.4.6 EREs Matching Multiple Characters**

6117 The following rules shall be used to construct EREs matching multiple characters from EREs
 6118 matching a single character:

- 6119 1. A concatenation of EREs shall match the concatenation of the character sequences
 6120 matched by each component of the ERE. A concatenation of EREs enclosed in parentheses
 6121 shall match whatever the concatenation without the parentheses matches. For example,
 6122 both the ERE "cd" and the ERE "(cd)" are matched by the third and fourth character of
 6123 the string "abcdefabcdef".
- 6124 2. When an ERE matching a single character or an ERE enclosed in parentheses is followed
 6125 by the special character <plus-sign> ('+'), together with that <plus-sign> it shall match
 6126 what one or more consecutive occurrences of the ERE would match. For example, the
 6127 ERE "b+(bc)" matches the fourth to seventh characters in the string "acabbbbcde".
 6128 And, "[ab]+" and "[ab][ab]*" are equivalent.
- 6129 3. When an ERE matching a single character or an ERE enclosed in parentheses is followed
 6130 by the special character <asterisk> ('*'), together with that <asterisk> it shall match
 6131 what zero or more consecutive occurrences of the ERE would match. For example, the
 6132 ERE "b*c" matches the first character in the string "cabbbbcde", and the ERE "b*cd"
 6133 matches the third to seventh characters in the string "cabbbbcdebbbbbbbcdbbc". And,
 6134 "[ab]*" and "[ab][ab]" are equivalent when matching the string "ab".
- 6135 4. When an ERE matching a single character or an ERE enclosed in parentheses is followed
 6136 by the special character <question-mark> ('?'), together with that <question-mark> it
 6137 shall match what zero or one consecutive occurrences of the ERE would match. For
 6138 example, the ERE "b?c" matches the second character in the string "acabbbbcde".
- 6139 5. When an ERE matching a single character or an ERE enclosed in parentheses is followed
 6140 by an interval expression of the format "{m}", "{m,}", or "{m,n}", together with that
 6141 interval expression it shall match what repeated consecutive occurrences of the ERE
 6142 would match. The values of *m* and *n* are decimal integers in the range 0
 6143 ≤ *m* ≤ *n* ≤ [RE_DUP_MAX], where *m* specifies the exact or minimum number of occurrences
 6144 and *n* specifies the maximum number of occurrences. The expression "{m}" matches
 6145 exactly *m* occurrences of the preceding ERE, "{m,}" matches at least *m* occurrences, and
 6146 "{m,n}" matches any number of occurrences between *m* and *n*, inclusive.

For example, in the string "abababcccccccd" the ERE "c{3}" is matched by characters seven to nine and the ERE "(ab){2,}" is matched by characters one to six.

The behavior of multiple adjacent duplication symbols ('+', '*', '?', and intervals) produces undefined results.

An ERE matching a single character repeated by an '*', '?', or an interval expression shall not match a null expression unless this is the only match for the repetition or it is necessary to satisfy the exact or minimum number of occurrences for the interval expression.

9.4.7 ERE Alternation

Two EREs separated by the special character <vertical-line> ('|') shall match a string that is matched by either. For example, the ERE "a((bc)|d)" matches the string "abc" and the string "ad". Single characters, or expressions matching single characters, separated by the <vertical-line> and enclosed in parentheses, shall be treated as an ERE matching a single character.

9.4.8 ERE Precedence

The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
Collation-related bracket symbols	[==] [::] [..]
Escaped characters	\<special character>
Bracket expression	[]
Grouping	()
Single-character-ERE duplication	* + ? {m,n}
Concatenation	
Anchoring	^ \$
Alternation	

For example, the ERE "abba|cde" matches either the string "abba" or the string "cde" (rather than the string "abbade" or "abbcde", because concatenation has a higher order of precedence than alternation).

9.4.9 ERE Expression Anchoring

An ERE can be limited to matching strings that begin or end a line; this is called "anchoring". The <circumflex> and <dollar-sign> special characters shall be considered ERE anchors when used anywhere outside a bracket expression. This shall have the following effects:

1. A <circumflex> ('^') outside a bracket expression shall anchor the expression or subexpression it begins to the beginning of a string; such an expression or subexpression can match only a sequence starting at the first character of a string. For example, the EREs "^ab" and "(^ab)" match "ab" in the string "abcdef", but fail to match in the string "cdefab", and the ERE "a^b" is valid, but can never match because the 'a' prevents the expression "^b" from matching starting at the first character.
2. A <dollar-sign> ('\$') outside a bracket expression shall anchor the expression or subexpression it ends to the end of a string; such an expression or subexpression can match only a sequence ending at the last character of a string. For example, the EREs "ef\$" and "(ef\$)" match "ef" in the string "abcdef", but fail to match in the string

*Regular Expressions**Extended Regular Expressions*

6188 "cdefab", and the ERE "e\$*f*" is valid, but can never match because the '*f*' prevents
 6189 the expression "e\$" from matching ending at the last character.

6190 9.5 Regular Expression Grammar

6191 Grammars describing the syntax of both basic and extended regular expressions are presented in
 6192 this section. The grammar takes precedence over the text. See XCU [Section 1.3](#) (on page 2287).

6193 9.5.1 BRE/ERE Grammar Lexical Conventions

6194 The lexical conventions for regular expressions are as described in this section.

6195 Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

6196 The following tokens are processed (in addition to those string constants shown in the
 6197 grammar):

6198	COLL_ELEM_SINGLE	Any single-character collating element, unless it is a META_CHAR .
6199	COLL_ELEM_MULTI	Any multi-character collating element.
6200	BACKREF	Applicable only to basic regular expressions. The character string 6201 consisting of a <backslash> character followed by a single-digit 6202 numeral, '1' to '9'.
6203	DUP_COUNT	Represents a numeric constant. It shall be an integer in the range 0 6204 ≤ DUP_COUNT ≤ RE_DUP_MAX . This token is only recognized 6205 when the context of the grammar requires it. At all other times, digits 6206 not preceded by a <backslash> character are treated as ORD_CHAR .
6207	META_CHAR	One of the characters:
6208	^	When found first in a bracket expression
6209	~	When found anywhere but first (after an initial '^', if any)
6210		or last in a bracket expression, or as the ending range point
6211		in a range expression
6212	]	When found anywhere but first (after an initial '^', if any)
6213		in a bracket expression
6214	L_ANCHOR	Applicable only to basic regular expressions. The character '^' 6215 when it appears as the first character of a basic regular expression 6216 and when not QUOTED_CHAR . The '^' may be recognized as an 6217 anchor elsewhere; see Section 9.3.8 (on page 187).
6218	ORD_CHAR	A character, other than one of the special characters in SPEC_CHAR .
6219	QUOTED_CHAR	In a BRE, one of the character sequences:
6220		\^ \. * \[\ \$ \\
6221		In an ERE, one of the character sequences:
6222		\^ \. \[\ \$ \ (\) \
6223		* \+ \? \{ \}

6224	R_ANCHOR	(Applicable only to basic regular expressions.) The character '\$' when it appears as the last character of a basic regular expression and when not QUOTED_CHAR . The '\$' may be recognized as an anchor elsewhere; see Section 9.3.8 (on page 187).
6225		
6226		
6227		
6228	SPEC_CHAR	For basic regular expressions, one of the following special characters:
6229	.	Anywhere outside bracket expressions
6230	\	Anywhere outside bracket expressions
6231	[	Anywhere outside bracket expressions
6232	^	When used as an anchor (see Section 9.3.8 , on page 187) or when first in a bracket expression
6233		
6234	\$	When used as an anchor
6235	*	Anywhere except first in an entire RE, anywhere in a bracket expression, directly following "\(", directly following an anchoring '^'
6236		
6237		
6238		For extended regular expressions, shall be one of the following special characters found anywhere outside bracket expressions:
6239		
6240	^ . [\$ ()	
6241	* + ? { }	
6242		(The close-parenthesis shall be considered special in this context only if matched with a preceding open-parenthesis.)
6243		

6244 9.5.2 RE and Bracket Expression Grammar

6245 This section presents the grammar for basic regular expressions, including the bracket
6246 expression grammar that is common to both BREs and EREs.

```

6247 %token    ORD_CHAR QUOTED_CHAR DUP_COUNT
6248 %token    BACKREF L_ANCHOR R_ANCHOR
6249 %token    Back_open_paren Back_close_paren
6250 /*      '('      ')'      */
6251 %token    Back_open_brace Back_close_brace
6252 /*      '{'      '}'      */
6253 /* The following tokens are for the Bracket Expression
6254    grammar common to both REs and EREs. */
6255 %token    COLL_ELEM_SINGLE COLL_ELEM_MULTI META_CHAR
6256 %token    Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close
6257 /*      '['      '='      '['      '.'      ':'      ':'      */
6258 %token    class_name
6259 /* class_name is a keyword to the LC_CTYPE locale category */
6260 /* (representing a character class) in the current locale */
6261 /* and is only recognized between [: and :] */
6262 %start    basic_reg_exp
6263 %%

```

*Regular Expressions**Regular Expression Grammar*

```

6264      /* -----
6265         Basic Regular Expression
6266      -----
6267      */
6268      basic_reg_exp : RE_expression
6269                    | L_ANCHOR
6270                    | R_ANCHOR
6271                    | L_ANCHOR R_ANCHOR
6272                    | L_ANCHOR RE_expression
6273                    | RE_expression R_ANCHOR
6274                    | L_ANCHOR RE_expression R_ANCHOR
6275                    ;
6276      RE_expression : simple_RE
6277                    | RE_expression simple_RE
6278                    ;
6279      simple_RE : nondupl_RE
6280                | nondupl_RE RE_dupl_symbol
6281                ;
6282      nondupl_RE : one_char_or_coll_elem_RE
6283                 | Back_open_paren RE_expression Back_close_paren
6284                 | BACKREF
6285                 ;
6286      one_char_or_coll_elem_RE : ORD_CHAR
6287                               | QUOTED_CHAR
6288                               | '.'
6289                               | bracket_expression
6290                               ;
6291      RE_dupl_symbol : '*'
6292                    | Back_open_brace DUP_COUNT Back_close_brace
6293                    | Back_open_brace DUP_COUNT ',' Back_close_brace
6294                    | Back_open_brace DUP_COUNT ',' DUP_COUNT Back_close_brace
6295                    ;
6296      /* -----
6297         Bracket Expression
6298      -----
6299      */
6300      bracket_expression : '[' matching_list ']'
6301                        | '[' nonmatching_list ']'
6302                        ;
6303      matching_list : bracket_list
6304                   ;
6305      nonmatching_list : '^' bracket_list
6306                      ;
6307      bracket_list : follow_list
6308                  | follow_list '-'
6309                  ;
6310      follow_list : expression_term
6311                 | follow_list expression_term
6312                 ;
6313      expression_term : single_expression
6314                     | range_expression
6315                     ;

```

```

6316     single_expression : end_range
6317                         | character_class
6318                         | equivalence_class
6319                         ;
6320     range_expression : start_range end_range
6321                     | start_range '-'
6322                     ;
6323     start_range      : end_range '-'
6324                     ;
6325     end_range        : COLL_ELEM_SINGLE
6326                     | collating_symbol
6327                     ;
6328     collating_symbol : Open_dot COLL_ELEM_SINGLE Dot_close
6329                     | Open_dot COLL_ELEM_MULTI Dot_close
6330                     | Open_dot META_CHAR Dot_close
6331                     ;
6332     equivalence_class : Open_equal COLL_ELEM_SINGLE Equal_close
6333                     | Open_equal COLL_ELEM_MULTI Equal_close
6334                     ;
6335     character_class : Open_colon class_name Colon_close
6336                     ;

```

6337 The BRE grammar does not permit **L_ANCHOR** or **R_ANCHOR** inside "**** (" and "**** " (which
6338 implies that '**^**' and '**\$**' are ordinary characters). This reflects the semantic limits on the
6339 application, as noted in [Section 9.3.8](#) (on page 187). Implementations are permitted to extend the
6340 language to interpret '**^**' and '**\$**' as anchors in these locations, and as such, conforming
6341 applications cannot use unescaped '**^**' and '**\$**' in positions inside "**** (" and "**** " that might
6342 be interpreted as anchors.

6343 9.5.3 ERE Grammar

6344 This section presents the grammar for extended regular expressions, excluding the bracket
6345 expression grammar.

6346 **Note:** The bracket expression grammar and the associated **%token** lines are identical between BREs
6347 and EREs. It has been omitted from the ERE section to avoid unnecessary editorial duplication.

```

6348 %token ORD_CHAR QUOTED_CHAR DUP_COUNT
6349 %start extended_reg_exp
6350 %%
6351 /* -----
6352    Extended Regular Expression
6353    ----- */
6354
6355 extended_reg_exp : ERE_branch
6356                 | extended_reg_exp '|' ERE_branch
6357                 ;
6358 ERE_branch      : ERE_expression
6359                 | ERE_branch ERE_expression
6360                 ;
6361 ERE_expression : one_char_or_coll_elem_ERE
6362                 | '^'
6363                 | '$'

```

*Regular Expressions**Regular Expression Grammar*

```

6364         | '(' extended_reg_exp ')'
6365         | ERE_expression ERE_dupl_symbol
6366         ;
6367 one_char_or_coll_elem_ERE : ORD_CHAR
6368         | QUOTED_CHAR
6369         | '.'
6370         | bracket_expression
6371         ;
6372 ERE_dupl_symbol : '*'
6373         | '+'
6374         | '?'
6375         | '{' DUP_COUNT '}'
6376         | '{' DUP_COUNT ',' '}'
6377         | '{' DUP_COUNT ',' DUP_COUNT '}'
6378         ;

```

The ERE grammar does not permit several constructs that previous sections specify as having undefined results:

- **ORD_CHAR** preceded by a <backslash> character
- One or more *ERE_dupl_symbols* appearing first in an ERE, or immediately following ' | ', '^ ', or ' ('
- ' { ' not part of a valid *ERE_dupl_symbol*
- ' | ' appearing first or last in an ERE, or immediately following ' | ' or ' (', or immediately preceding ') '

Implementations are permitted to extend the language to allow these. Conforming applications cannot use such constructs.

(blank page)

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

*Directory Structure and Devices***10.1 Directory Structure and Files**

The following directories shall exist on conforming systems and conforming applications shall make use of them only as described. Strictly conforming applications shall not assume the ability to create files in any of these directories, unless specified below.

/ The root directory.

/dev Contains **/dev/console**, **/dev/null**, and **/dev/tty**, described below.

The following directory shall exist on conforming systems and shall be used as described:

/tmp A directory made available for applications that need a place to create temporary files. Applications shall be allowed to create files in this directory, but shall not assume that such files are preserved between invocations of the application.

The following files shall exist on conforming systems and shall be both readable and writable:

/dev/null An infinite data source and data sink. Data written to **/dev/null** shall be discarded. Reads from **/dev/null** shall always return end-of-file (EOF).

/dev/tty In each process, a synonym for the controlling terminal associated with the process group of that process, if any. It is useful for programs or shell procedures that wish to be sure of writing messages to or reading data from the terminal no matter how output has been redirected. It can also be used for applications that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

The following file shall exist on conforming systems and need not be readable or writable:

/dev/console The **/dev/console** file is a generic name given to the system console (see [Section 3.384](#), on page 95). It is usually linked to an implementation-defined special file. It shall provide an interface to the system console conforming to the requirements of [Chapter 11](#) (on page 199).

10.2 Output Devices and Terminal Types

The utilities in the Shell and Utilities volume of POSIX.1-2008 historically have been implemented on a wide range of terminal types, but a conforming implementation need not support all features of all utilities on every conceivable terminal. POSIX.1-2008 states which features are optional for certain classes of terminals in the individual utility description sections. The implementation shall document in the system documentation which terminal types it supports and which of these features and utilities are not supported by each terminal.

When a feature or utility is not supported on a specific terminal type, as allowed by POSIX.1-2008, and the implementation considers such a condition to be an error preventing use of the feature or utility, the implementation shall indicate such conditions through diagnostic messages or exit status values or both (as appropriate to the specific utility description) that inform the user that the terminal type lacks the appropriate capability.

POSIX.1-2008 uses a notational convention based on historical practice that identifies some of the control characters defined in Section 7.3.1 (on page 139) in a manner easily remembered by users on many terminals. The correspondence between this “<control>-char” notation and the actual control characters is shown in the following table. When POSIX.1-2008 refers to a character by its <control>-name, it is referring to the actual control character shown in the Value column of the table, which is not necessarily the exact control key sequence on all terminals. Some terminals have keyboards that do not allow the direct transmission of all the non-alphanumeric characters shown. In such cases, the system documentation shall describe which data sequences transmitted by the terminal are interpreted by the system as representing the special characters.

Table 10-1 Control Character Names

Name	Value	Symbolic Name	Name	Value	Symbolic Name
<control>-A	<SOH>	<SOH>	<control>-Q	<DC1>	<DC1>
<control>-B	<STX>	<STX>	<control>-R	<DC2>	<DC2>
<control>-C	<ETX>	<ETX>	<control>-S	<DC3>	<DC3>
<control>-D	<EOT>	<EOT>	<control>-T	<DC4>	<DC4>
<control>-E	<ENQ>	<ENQ>	<control>-U	<NAK>	<NAK>
<control>-F	<ACK>	<ACK>	<control>-V	<SYN>	<SYN>
<control>-G	<BEL>	<alert>	<control>-W	<ETB>	<ETB>
<control>-H	<BS>	<backspace>	<control>-X	<CAN>	<CAN>
<control>-I	<HT>	<tab>	<control>-Y		
<control>-J	<LF>	<linefeed>	<control>-Z	<SUB>	<SUB>
<control>-K	<VT>	<vertical-tab>	<control>-[	<ESC>	<ESC>
<control>-L	<FF>	<form-feed>	<control>-\	<FS>	<FS>
<control>-M	<CR>	<carriage-return>	<control>-]	<GS>	<GS>
<control>-N	<SO>	<SO>	<control>-^	<RS>	<RS>
<control>-O	<SI>	<SI>	<control>-_	<US>	<US>
<control>-P	<DLE>	<DLE>	<control>-?	<DEL>	<DEL>

Note: The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that the keystrokes represent control-shift-letter sequences.

General Terminal Interface

This chapter describes a general terminal interface that shall be provided. It shall be supported on any asynchronous communications ports if the implementation provides them. It is implementation-defined whether it supports network connections or synchronous ports, or both.

11.1 Interface Characteristics

11.1.1 Opening a Terminal Device File

When a terminal device file is opened, it normally causes the thread to wait until a connection is established. In practice, application programs seldom open these files; they are opened by special programs and become an application's standard input, output, and error files.

Cases where applications do open a terminal device are as follows:

1. Opening `/dev/tty`, or the pathname returned by `ctermid()`, in order to obtain a file descriptor for the controlling terminal; see [Section 11.1.3](#) (on page 200).
2. Opening the slave side of a pseudo-terminal; see XSH [ptsname\(\)](#).
3. Opening a modem or similar piece of equipment connected by a serial line. In this case, the terminal parameters (see [Section 11.2](#), on page 205) may be initialized to default settings by the implementation in between the last close of the device by any process and the next open of the device, or they may persist from one use to the next. The terminal parameters can be set to values that ensure the terminal behaves in a conforming manner by means of the `O_TTY_INIT` open flag when opening a terminal device that is not already open in any process, or by executing the `stty` utility with the operand `sane`.

As described in `open()`, opening a terminal device file with the `O_NONBLOCK` flag clear shall cause the thread to block until the terminal device is ready and available. If `CLOCAL` mode is not set, this means blocking until a connection is established. If `CLOCAL` mode is set in the terminal, or the `O_NONBLOCK` flag is specified in the `open()`, the `open()` function shall return a file descriptor without waiting for a connection to be established.

11.1.2 Process Groups

A terminal may have a foreground process group associated with it. This foreground process group plays a special role in handling signal-generating input characters, as discussed in [Section 11.1.9](#) (on page 203).

A command interpreter process supporting job control can allocate the terminal to different jobs, or process groups, by placing related processes in a single process group and associating this process group with the terminal. A terminal's foreground process group may be set or examined by a process, assuming the permission requirements are met; see *tcgetpgrp()* and *tcsetpgrp()*. The terminal interface aids in this allocation by restricting access to the terminal by processes that are not in the current process group; see [Section 11.1.4](#) (on page 201).

When there is no longer any process whose process ID or process group ID matches the foreground process group ID, the terminal shall have no foreground process group. It is unspecified whether the terminal has a foreground process group when there is a process whose process ID matches the foreground process group ID, but whose process group ID does not. No actions defined in POSIX.1-2008, other than allocation of a controlling terminal or a successful call to *tcsetpgrp()*, shall cause a process group to become the foreground process group of the terminal.

11.1.3 The Controlling Terminal

A terminal may belong to a process as its controlling terminal. Each process of a session that has a controlling terminal has the same controlling terminal. A terminal may be the controlling terminal for at most one session. The controlling terminal for a session is allocated by the session leader in an implementation-defined manner. If a session leader has no controlling terminal, and opens a terminal device file that is not already associated with a session without using the *O_NOCTTY* option (see *open()*), it is implementation-defined whether the terminal becomes the controlling terminal of the session leader. If a process which is not a session leader opens a terminal file, or the *O_NOCTTY* option is used on *open()*, then that terminal shall not become the controlling terminal of the calling process. When a controlling terminal becomes associated with a session, its foreground process group shall be set to the process group of the session leader.

The controlling terminal is inherited by a child process during a *fork()* function call. A process relinquishes its controlling terminal when it creates a new session with the *setsid()* function; other processes remaining in the old session that had this terminal as their controlling terminal continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in the current session) associated with the controlling terminal, it is unspecified whether all processes that had that terminal as their controlling terminal cease to have any controlling terminal. Whether and how a session leader can reacquire a controlling terminal after the controlling terminal has been relinquished in this fashion is unspecified. A process does not relinquish its controlling terminal simply by closing all of its file descriptors associated with the controlling terminal if other processes continue to have it open.

When a controlling process terminates, the controlling terminal is dissociated from the current session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by other processes in the earlier session may be denied, with attempts to access the terminal treated as if a modem disconnect had been sensed.

11.1.4 Terminal Access Control

If a process is in the foreground process group of its controlling terminal, read operations shall be allowed, as described in [Section 11.1.5](#). Any attempts by a process in a background process group to read from its controlling terminal cause its process group to be sent a SIGTTIN signal unless one of the following special cases applies: if the reading process is ignoring or blocking the SIGTTIN signal, or if the process group of the reading process is orphaned, the *read()* shall return -1 , with *errno* set to [EIO] and no signal shall be sent. The default action of the SIGTTIN signal shall be to stop the process to which it is sent. See [<signal.h>](#).

If a process is in the foreground process group of its controlling terminal, write operations shall be allowed as described in [Section 11.1.8](#) (on page 203). Attempts by a process in a background process group to write to its controlling terminal shall cause the process group to be sent a SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if TOSTOP is set and the process is ignoring or blocking the SIGTTOU signal, the process is allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the process group of the writing process is orphaned, and the writing process is not ignoring or blocking the SIGTTOU signal, the *write()* shall return -1 , with *errno* set to [EIO] and no signal shall be sent.

Certain calls that set terminal parameters are treated in the same fashion as *write()*, except that TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set (see [Section 11.2.5](#) (on page 210), *tcdrain()*, *tcflow()*, *tcflush()*, *tcsendbreak()*, *tcsetattr()*, and *tcsetpgrp()*).

11.1.5 Input Processing and Reading Data

A terminal device associated with a terminal device file may operate in full-duplex mode, so that data may arrive even while output is occurring. Each terminal device file has an input queue associated with it, into which incoming data is stored by the system before being read by a process. The system may impose a limit, {MAX_INPUT}, on the number of bytes that may be stored in the input queue. The behavior of the system when this limit is exceeded is implementation-defined.

Two general kinds of input processing are available, determined by whether the terminal device file is in canonical mode or non-canonical mode. These modes are described in [Section 11.1.6](#) (on page 202) and [Section 11.1.7](#) (on page 202). Additionally, input characters are processed according to the *e_iflag* (see [Section 11.2.2](#), on page 206) and *c_lflag* (see [Section 11.2.5](#), on page 210) fields. Such processing can include “echoing”, which in general means transmitting input characters immediately back to the terminal when they are received from the terminal. This is useful for terminals that can operate in full-duplex mode.

The manner in which data is provided to a process reading from a terminal device file is dependent on whether the terminal file is in canonical or non-canonical mode, and on whether or not the O_NONBLOCK flag is set by *open()* or *fcntl()*.

If the O_NONBLOCK flag is clear, then the read request shall be blocked until data is available or a signal has been received. If the O_NONBLOCK flag is set, then the read request shall be completed, without blocking, in one of three ways:

1. If there is enough data available to satisfy the entire request, the *read()* shall complete successfully and shall return the number of bytes read.
2. If there is not enough data available to satisfy the entire request, the *read()* shall complete successfully, having read as much data as possible, and shall return the number of bytes it was able to read.

3. If there is no data available, the *read()* shall return -1 , with *errno* set to [EAGAIN].

When data is available depends on whether the input processing mode is canonical or non-canonical. [Section 11.1.6](#) and [Section 11.1.7](#) describe each of these input processing modes.

11.1.6 Canonical Mode Input Processing

In canonical mode input processing, terminal input is processed in units of lines. A line is delimited by a <newline> character (NL), an end-of-file character (EOF), or an end-of-line (EOL) character. See [Section 11.1.9](#) (on page 203) for more information on EOF and EOL. This means that a read request shall not return until an entire line has been typed or a signal has been received. Also, no matter how many bytes are requested in the *read()* call, at most one line shall be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even one, may be requested in a *read()* without losing information.

If {MAX_CANON} is defined for this terminal device, it shall be a limit on the number of bytes in a line. The behavior of the system when this limit is exceeded is implementation-defined. If {MAX_CANON} is not defined, there shall be no such limit; see *pathconf()*.

Erase and kill processing occur when either of two special characters, the ERASE and KILL characters (see [Section 11.1.9](#), on page 203), is received. This processing shall affect data in the input queue that has not yet been delimited by an NL, EOF, or EOL character. This un-delimited data makes up the current line. The ERASE character shall delete the last character in the current line, if there is one. The KILL character shall delete all data in the current line, if there is any. The ERASE and KILL characters shall have no effect if there is no data in the current line. The ERASE and KILL characters themselves shall not be placed in the input queue.

11.1.7 Non-Canonical Mode Input Processing

In non-canonical mode input processing, input bytes are not assembled into lines, and erase and kill processing shall not occur. The values of the MIN and TIME members of the *c_cc* array are used to determine how to process the bytes received. POSIX.1-2008 does not specify whether the setting of O_NONBLOCK takes precedence over MIN or TIME settings. Therefore, if O_NONBLOCK is set, *read()* may return immediately, regardless of the setting of MIN or TIME. Also, if no data is available, *read()* may either return 0, or return -1 with *errno* set to [EAGAIN].

MIN represents the minimum number of bytes that should be received when the *read()* function returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and short-term data transmissions. If MIN is greater than {MAX_INPUT}, the response to the request is undefined. The four possible values for MIN and TIME and their interactions are described below.

Case A: MIN>0, TIME>0

In case A, TIME serves as an inter-byte timer which shall be activated after the first byte is received. Since it is an inter-byte timer, it shall be reset after a byte is received. The interaction between MIN and TIME is as follows. As soon as one byte is received, the inter-byte timer shall be started. If MIN bytes are received before the inter-byte timer expires (remember that the timer is reset upon receipt of each byte), the read shall be satisfied. If the timer expires before MIN bytes are received, the characters received to that point shall be returned to the user. Note that if TIME expires at least one byte shall be returned because the timer would not have been enabled unless a byte was received. In this case (MIN>0, TIME>0) the read shall block until the MIN and TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is in the buffer at the time of the *read()*, the result shall be as if data has been received immediately

after the *read()*.

Case B: MIN>0, TIME=0

In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A pending read shall not be satisfied until MIN bytes are received (that is, the pending read shall block until MIN bytes are received), or a signal is received. A program that uses case B to read record-based terminal I/O may block indefinitely in the read operation.

Case C: MIN=0, TIME>0

In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read timer that shall be activated as soon as the *read()* function is processed. A read shall be satisfied as soon as a single byte is received or the read timer expires. Note that in case C if the timer expires, no bytes shall be returned. If the timer does not expire, the only way the read can be satisfied is if a byte is received. If bytes are not received, the read shall not block indefinitely waiting for a byte; if no byte is received within TIME*0.1 seconds after the read is initiated, the *read()* shall return a value of zero, having read no data. If data is in the buffer at the time of the *read()*, the timer shall be started as if data has been received immediately after the *read()*.

Case D: MIN=0, TIME=0

The minimum of either the number of bytes requested or the number of bytes currently available shall be returned without waiting for more bytes to be input. If no characters are available, *read()* shall return a value of zero, having read no data.

11.1.8 Writing Data and Output Processing

When a process writes one or more bytes to a terminal device file, they are processed according to the *c_oflag* field (see [Section 11.2.3](#), on page 207). The implementation may provide a buffering mechanism; as such, when a call to *write()* completes, all of the bytes written have been scheduled for transmission to the device, but the transmission has not necessarily completed. See *write()* for the effects of O_NONBLOCK on *write()*.

11.1.9 Special Characters

Certain characters have special functions on input or output or both. These functions are summarized as follows:

INTR Special character on input, which is recognized if the ISIG flag is set. Generates a SIGINT signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the INTR character shall be discarded when processed.

QUIT Special character on input, which is recognized if the ISIG flag is set. Generates a SIGQUIT signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the QUIT character shall be discarded when processed.

ERASE Special character on input, which is recognized if the ICANON flag is set. Erases the last character in the current line; see [Section 11.1.6](#) (on page 202). It shall not erase beyond the start of a line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the ERASE character shall be discarded when processed.

Interface Characteristics

General Terminal Interface

6657	KILL	Special character on input, which is recognized if the ICANON flag is set. Deletes the entire line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the KILL character shall be discarded when processed.
6658		
6659		
6660	EOF	Special character on input, which is recognized if the ICANON flag is set. When received, all the bytes waiting to be read are immediately passed to the process without waiting for a <newline>, and the EOF is discarded. Thus, if there are no bytes waiting (that is, the EOF occurred at the beginning of a line), a byte count of zero shall be returned from the <i>read()</i> , representing an end-of-file indication. If ICANON is set, the EOF character shall be discarded when processed.
6661		
6662		
6663		
6664		
6665		
6666	NL	Special character on input, which is recognized if the ICANON flag is set. It is the line delimiter <newline>. It cannot be changed.
6667		
6668	EOL	Special character on input, which is recognized if the ICANON flag is set. It is an additional line delimiter, like NL.
6669		
6670	SUSP	If the ISIG flag is set, receipt of the SUSP character shall cause a SIGTSTP signal to be sent to all processes in the foreground process group for which the terminal is the controlling terminal, and the SUSP character shall be discarded when processed.
6671		
6672		
6673	STOP	Special character on both input and output, which is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to suspend output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. If IXON is set, the STOP character shall be discarded when processed.
6674		
6675		
6676		
6677	START	Special character on both input and output, which is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to resume output that has been suspended by a STOP character. If IXON is set, the START character shall be discarded when processed.
6678		
6679		
6680		
6681	CR	Special character on input, which is recognized if the ICANON flag is set; it is the carriage-return character. When ICANON and ICRNL are set and IGNCR is not set, this character shall be translated into an NL, and shall have the same effect as an NL character.
6682		
6683		
6684		
6685		The NL and CR characters cannot be changed. It is implementation-defined whether the START and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and SUSP shall be changeable to suit individual tastes. Special character functions associated with changeable special control characters can be disabled individually.
6686		
6687		
6688		
6689		If two or more special characters have the same value, the function performed when that character is received is undefined.
6690		
6691		A special character is recognized not only by its value, but also by its context; for example, an implementation may support multi-byte sequences that have a meaning different from the meaning of the bytes when considered individually. Implementations may also support additional single-byte functions. These implementation-defined multi-byte or single-byte functions shall be recognized only if the IEXTEN flag is set; otherwise, data is received without interpretation, except as required to recognize the special characters defined in this section.
6692		
6693		
6694		
6695		
6696		
6697	XSI	If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding <backslash> character, in which case no special function shall occur.
6698		

11.1.10 Modem Disconnect

If a modem disconnect is detected by the terminal interface for a controlling terminal, and if CLOCAL is not set in the *c_cflag* field for the terminal (see Section 11.2.4, on page 209), the SIGHUP signal shall be sent to the controlling process for which the terminal is the controlling terminal. Unless other arrangements have been made, this shall cause the controlling process to terminate (see *exit()*). Any subsequent read from the terminal device shall return the value of zero, indicating end-of-file; see *read()*. Thus, processes that read a terminal file and test for end-of-file can terminate appropriately after a disconnect. If the EIO condition as specified in *read()* also exists, it is unspecified whether on EOF condition or [EIO] is returned. Any subsequent *write()* to the terminal device shall return -1, with *errno* set to [EIO], until the device is closed.

11.1.11 Closing a Terminal Device File

The last process to close a terminal device file shall cause any output to be sent to the device and any input to be discarded. If HUPCL is set in the control modes and the communications port supports a disconnect function, the terminal device shall perform a disconnect.

11.2 Parameters that Can be Set

11.2.1 The **termios** Structure

Routines that need to control certain terminal I/O characteristics shall do so by using the **termios** structure as defined in the `<termios.h>` header.

Since the **termios** structure may include additional members, and the standard members may include both standard and non-standard modes, the structure should never be initialized directly by the application as this may cause the terminal to behave in a non-conforming manner. When opening a terminal device (other than a pseudo-terminal) that is not already open in any process, it should be opened with the O_TTY_INIT flag before initializing the structure using *tcgetattr()* to ensure that any non-standard elements of the **termios** structure are set to values that result in conforming behavior of the terminal interface.

The members of the **termios** structure include (but are not limited to):

Member Type	Array Size	Member Name	Description
tcflag_t	NCCS	<i>c_iflag</i>	Input modes.
tcflag_t		<i>c_oflag</i>	Output modes.
tcflag_t		<i>c_cflag</i>	Control modes.
tcflag_t		<i>c_lflag</i>	Local modes.
cc_t		<i>c_cc[]</i>	Control characters.

The **tcflag_t** and **cc_t** types are defined in the `<termios.h>` header. They shall be unsigned integer types.

11.2.2 Input Modes

Values of the *c_iflag* field describe the basic terminal input control, and are composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in [<termios.h>](#):

Mask Name	Description
BRKINT	Signal interrupt on break.
ICRNL	Map CR to NL on input.
IGNBRK	Ignore break condition.
IGNCR	Ignore CR.
IGNPAR	Ignore characters with parity errors.
INLCR	Map NL to CR on input.
INPCK	Enable input parity check.
ISTRIP	Strip character.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IXON	Enable start/stop output control.
PARMRK	Mark parity errors.

In the context of asynchronous serial data transmission, a break condition shall be defined as a sequence of zero-valued bits that continues for more than the time to send one byte. The entire sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a time equivalent to more than one byte. In contexts other than asynchronous serial data transmission, the definition of a break condition is implementation-defined.

If IGNBRK is set, a break condition detected on input shall be ignored; that is, not put on the input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break condition shall flush the input and output queues, and if the terminal is the controlling terminal of a foreground process group, the break condition shall generate a single SIGINT signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break condition shall be read as a single 0x00, or if PARMRK is set, as 0xff 0x00 0x00.

If IGNPAR is set, a byte with a framing or parity error (other than break) shall be ignored.

If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than break) shall be given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is a two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) shall be given to the application as a single byte 0x00.

If INPCK is set, input parity checking shall be enabled. If INPCK is not set, input parity checking shall be disabled, allowing output parity generation without input parity errors. Note that whether input parity checking is enabled or disabled is independent of whether parity detection is enabled or disabled (see [Section 11.2.4](#), on page 209). If parity detection is enabled but input parity checking is disabled, the hardware to which the terminal is connected shall recognize the parity bit, but the terminal special file shall not check whether or not this bit is correctly set.

If ISTRIP is set, valid input bytes shall first be stripped to seven bits; otherwise, all eight bits shall be processed.

If INLCR is set, a received NL character shall be translated into a CR character. If IGNCR is set, a received CR character shall be ignored (not read). If IGNCR is not set and ICRNL is set, a received CR character shall be translated into an NL character.

If IXANY is set, any input character shall restart output that has been suspended.

*General Terminal Interface**Parameters that Can be Set*

6781 If IXON is set, start/stop output control shall be enabled. A received STOP character shall
 6782 suspend output and a received START character shall restart output. When IXON is set, START
 6783 and STOP characters are not read, but merely perform flow control functions. When IXON is not
 6784 set, the START and STOP characters shall be read.

6785 If IXOFF is set, start/stop input control shall be enabled. The system shall transmit STOP
 6786 characters, which are intended to cause the terminal device to stop transmitting data, as needed
 6787 to prevent the input queue from overflowing and causing implementation-defined behavior,
 6788 and shall transmit START characters, which are intended to cause the terminal device to resume
 6789 transmitting data, as soon as the device can continue transmitting data without risk of
 6790 overflowing the input queue. The precise conditions under which STOP and START characters
 6791 are transmitted are implementation-defined.

6792 The initial input control value after *open()* is implementation-defined.

6793 11.2.3 Output Modes

6794 The *c_oflag* field specifies the terminal interface's treatment of output, and is composed of the
 6795 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
 6796 symbols in the following table are defined in `<termios.h>`:

Mask Name	Description
OPOST	Perform output processing.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select newline delays:
NL0	Newline character type 0.
NL1	Newline character type 1.
CRDLY	Select carriage-return delays:
CR0	Carriage-return delay type 0.
CR1	Carriage-return delay type 1.
CR2	Carriage-return delay type 2.
CR3	Carriage-return delay type 3.
TABDLY	Select horizontal-tab delays:
TAB0	Horizontal-tab delay type 0.
TAB1	Horizontal-tab delay type 1.
TAB2	Horizontal-tab delay type 2.
TAB3	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	Backspace-delay type 0.
BS1	Backspace-delay type 1.
VTDLY	Select vertical-tab delays:
VT0	Vertical-tab delay type 0.
VT1	Vertical-tab delay type 1.
FFDLY	Select form-feed delays:
FF0	Form-feed delay type 0.
FF1	Form-feed delay type 1.

If OPOST is set, output data shall be post-processed as described below, so that lines of text are modified to appear appropriately on the terminal device; otherwise, characters shall be transmitted without change.

If ONLCR is set, the NL character shall be transmitted as the CR-NL character pair. If OCRNL is set, the CR character shall be transmitted as the NL character. If ONOCR is set, no CR character shall be transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer shall be set to 0 and the delays specified for CR shall be used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer shall also be set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 shall indicate no delay. If OFILL is set, fill characters shall be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character shall be DEL; otherwise, NUL.

If a <form-feed> or <vertical-tab> delay is specified, it shall last for about 2 seconds.

Newline delay shall last about 0.10 seconds. If ONLRET is set, the carriage-return delays shall be used instead of the newline delays. If OFILL is set, two fill characters shall be transmitted.

Carriage-return delay type 1 shall be dependent on the current column position, type 2 shall be

General Terminal Interface

Parameters that Can be Set

about 0.10 seconds, and type 3 shall be about 0.15 seconds. If OFILL is set, delay type 1 shall transmit two fill characters, and type 2 four fill characters.

Horizontal-tab delay type 1 shall be dependent on the current column position. Type 2 shall be about 0.10 seconds. Type 3 specifies that <tab> characters shall be expanded into <space> characters. If OFILL is set, two fill characters shall be transmitted for any delay.

Backspace delay shall last about 0.05 seconds. If OFILL is set, one fill character shall be transmitted.

The actual delays depend on line speed and system load.

The initial output control value after *open()* is implementation-defined.

11.2.4 Control Modes

The *c_cflag* field describes the hardware control of the terminal, and is composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in [<termios.h>](#); not all values specified are required to be supported by the underlying hardware (if any). If the terminal is a pseudo-terminal, it is unspecified whether non-default values are unsupported, or are supported and emulated in software, or are handled by *tcsetattr()*, *tcgetattr()*, and the *stty* utility as if they are supported but have no effect on the behavior of the terminal interface.

Mask Name	Description
CLOCAL	Ignore modem status lines.
CREAD	Enable receiver.
CSIZE	Number of bits transmitted or received per byte:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits.
CSTOPB	Send two stop bits, else one.
HUPCL	Hang up on last close.
PARENB	Parity enable.
PARODD	Odd parity, else even.

In addition, the input and output baud rates are stored in the **termios** structure. The symbols in the following table are defined in [<termios.h>](#). Not all values specified are required to be supported by the underlying hardware (if any). For pseudo-terminals, the input and output baud rates set in the **termios** structure need not affect the speed of data transmission through the terminal interface.

Note. The term “baud” is used historically here, but is not technically correct. This is properly “bits per second”, which may not be the same as baud. However, the term is used because of the historical usage and understanding.

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

The following functions are provided for getting and setting the values of the input and output baud rates in the **termios** structure: `cfgetispeed()`, `cfgetospeed()`, `cfsetispeed()`, and `cfsetospeed()`. The effects on the terminal device shall not become effective and not all errors need be detected until the `tcsetattr()` function is successfully called.

The CSIZE bits shall specify the number of transmitted or received bits per byte. If ISTRIP is not set, the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-order bits shall be zero, but the value of any other bits beyond CSIZE is unspecified when read. CSIZE shall not include the parity bit, if any. If CSTOPB is set, two stop bits shall be used; otherwise, one stop bit. For example, at 110 baud, two stop bits are normally used.

If CREAD is set, the receiver shall be enabled; otherwise, no characters shall be received.

If PARENB is set, parity generation and detection shall be enabled and a parity bit is added to each byte. If parity is enabled, PARODD shall specify odd parity if set; otherwise, even parity shall be used.

If HUPCL is set, the modem control lines for the port shall be lowered when the last process with the port open closes the port or the process terminates. The modem connection shall be broken.

If CLOCAL is set, a connection shall not depend on the state of the modem status lines. If CLOCAL is clear, the modem status lines shall be monitored.

Under normal circumstances, a call to the `open()` function shall wait for the modem connection to complete. However, if the `O_NONBLOCK` flag is set (see `open()`) or if CLOCAL has been set, the `open()` function shall return immediately without waiting for the connection.

If the object for which the control modes are set is not an asynchronous serial connection, some of the modes may be ignored; for example, if an attempt is made to set the baud rate on a network connection to a terminal on another host, the baud rate need not be set on the connection between that terminal and the machine to which it is directly connected.

The initial hardware control value after `open()` is implementation-defined.

11.2.5 Local Modes

The `c_lflag` field of the argument structure is used to control various functions. It is composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in [<termios.h>](#).

General Terminal Interface

Parameters that Can be Set

Mask Name	Description
ECHO	Enable echo.
ECHOE	Echo ERASE as an error correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo <newline>.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended (implementation-defined) functions.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt, quit, or suspend.
TOSTOP	Send SIGTTOU for background output.

If ECHO is set, input characters shall be echoed back to the terminal. If ECHO is clear, input characters shall not be echoed.

If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if possible, the last character in the current line from the display. If there is no character to erase, an implementation may echo an indication that this was the case, or do nothing.

If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the line from the display or shall echo the <newline> character after the KILL character.

If ECHONL and ICANON are set, the <newline> character shall be echoed even if ECHO is not set.

If ICANON is set, canonical processing shall be enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as described in [Section 11.1.6](#) (on page 202).

If ICANON is not set, read requests shall be satisfied directly from the input queue. A read shall not be satisfied until at least MIN bytes have been received or the timeout value TIME expired between bytes. The time value represents tenths of a second. See [Section 11.1.7](#) (on page 202) for more details.

If IEXTEN is set, implementation-defined functions shall be recognized from the input data. It is implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF. If IEXTEN is not set, implementation-defined functions shall not be recognized and the corresponding input characters are processed as described for ICANON, ISIG, IXON, and IXOFF.

If ISIG is set, each input character shall be checked against the special control characters INTR, QUIT, and SUSP. If an input character matches one of these control characters, the function associated with that character shall be performed. If ISIG is not set, no checking shall be done. Thus these special input functions are possible only if ISIG is set.

If NOFLSH is set, the normal flush of the input and output queues associated with the INTR, QUIT, and SUSP characters shall not be done.

If TOSTOP is set, the signal SIGTTOU shall be sent to the process group of a process that tries to write to its controlling terminal if it is not in the foreground process group for that terminal. This signal, by default, stops the members of the process group. Otherwise, the output generated by that process shall be output to the current output stream. Processes that are blocking or ignoring SIGTTOU signals are excepted and allowed to produce output, and the SIGTTOU signal shall not be sent.

The initial local control value after *open()* is implementation-defined.

11.2.6 Special Control Characters

The special control character values shall be defined by the array `c_cc`. The subscript name and description for each element in both canonical and non-canonical modes are as follows:

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF	VINTR	EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR		INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

The subscript values are unique, except that the VMIN and VTIME subscripts may have the same values as the VEOF and VEOL subscripts, respectively.

Implementations that do not support changing the START and STOP characters may ignore the character values in the `c_cc` array indexed by the VSTART and VSTOP subscripts when `tcsetattr()` is called, but shall return the value in use when `tcgetattr()` is called.

The initial values of all control characters are implementation-defined.

If the value of one of the changeable special control characters (see [Section 11.1.9](#), on page 203) is `_POSIX_VDISABLE`, that function shall be disabled; that is, no input data is recognized as the disabled special character. If ICANON is not set, the value of `_POSIX_VDISABLE` has no special meaning for the VMIN and VTIME entries of the `c_cc` array.

12.1 Utility Argument Syntax

This section describes the argument syntax of the standard utilities and introduces terminology used throughout POSIX.1-2008 for describing the arguments processed by the utilities.

Within POSIX.1-2008, a special notation is used for describing the syntax of a utility's arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated by this example (see XCU [Section 2.9.1](#), on page 2316):

```
utility_name[-a] [-b] [-c option_argument]
          [-d|-e] [-f[option_argument]] [operand...]
```

The notation used for the SYNOPSIS sections imposes requirements on the implementors of the standard utilities and provides a simple reference for the application developer or system user.

1. The utility in the example is named *utility_name*. It is followed by options, option-arguments, and operands. The arguments that consist of <hyphen> characters and single letters or digits, such as 'a', are known as "options" (or, historically, "flags"). Certain options are followed by an "option-argument", as shown with [-c *option_argument*]. The arguments following the last options and option-arguments are named "operands".
2. Option-arguments are shown separated from their options by <blank> characters, except when the option-argument is enclosed in the '[' and ']' notation to indicate that it is optional. This reflects the situation in which an optional option-argument (if present) is included within the same argument string as the option; for a mandatory option-argument, it is the next argument. The Utility Syntax Guidelines in [Section 12.2](#) (on page 215) require that the option be a separate argument from its option-argument and that option-arguments not be optional, but there are some exceptions in POSIX.1-2008 to ensure continued operation of historical applications:
 - a. If the SYNOPSIS of a standard utility shows an option with a mandatory option-argument (as with [-c *option_argument*] in the example), a conforming application shall use separate arguments for that option and its option-argument. However, a conforming implementation shall also permit applications to specify the option and option-argument in the same argument string without intervening <blank> characters.
 - b. If the SYNOPSIS shows an optional option-argument (as with [-f[*option_argument*]] in the example), a conforming application shall place any option-argument for that option directly adjacent to the option in the same argument string, without intervening <blank> characters. If the utility receives an argument containing only the option, it shall behave as specified in its description for an omitted option-argument; it shall not treat the next argument (if any) as the option-argument for that option.

3. Options are usually listed in alphabetical order unless this would make the utility description more confusing. There are no implied relationships between the options based upon the order in which they appear, unless otherwise stated in the OPTIONS section, or unless the exception in Guideline 11 of [Section 12.2](#) (on page 215) applies. If an option that does not have option-arguments is repeated, the results are undefined, unless otherwise stated.

4. Frequently, names of parameters that require substitution by actual values are shown with embedded <underscore> characters. Alternatively, parameters are shown as follows:

<parameter name>

The angle brackets are used for the symbolic grouping of a phrase representing a single parameter and conforming applications shall not include them in data submitted to the utility.

5. When a utility has only a few permissible options, they are sometimes shown individually, as in the example. Utilities with many flags generally show all of the individual flags (that do not take option-arguments) grouped, as in:

utility_name [-abcDxyz] [-p arg] [operand]

Utilities with very complex arguments may be shown as follows:

utility_name [options] [operands]

6. Unless otherwise specified, whenever an operand or option-argument is, or contains, a numeric value:

- The number is interpreted as a decimal integer.
- Numerals in the range 0 to 2 147 483 647 are syntactically recognized as numeric values.
- When the utility description states that it accepts negative numbers as operands or option-arguments, numerals in the range -2 147 483 647 to 2 147 483 647 are syntactically recognized as numeric values.
- Ranges greater than those listed here are allowed.

This does not mean that all numbers within the allowable range are necessarily semantically correct. A standard utility that accepts an option-argument or operand that is to be interpreted as a number, and for which a range of values smaller than that shown above is permitted by the POSIX.1-2008, describes that smaller range along with the description of the option-argument or operand. If an error is generated, the utility's diagnostic message shall indicate that the value is out of the supported range, not that it is syntactically incorrect.

7. Arguments or option-arguments enclosed in the ' [' and '] ' notation are optional and can be omitted. Conforming applications shall not include the ' [' and '] ' symbols in data submitted to the utility.

8. Arguments separated by the ' | ' (<vertical-line>) bar notation are mutually-exclusive. Conforming applications shall not include the ' | ' symbol in data submitted to the utility. Alternatively, mutually-exclusive options and operands may be listed with multiple synopsis lines.

Utility Conventions

Utility Argument Syntax

For example:

```
utility_name -d[-a] [-c option_argument] [operand...]
utility_name [-a] [-b] [operand...]
```

When multiple synopsis lines are given for a utility, it is an indication that the utility has mutually-exclusive arguments. These mutually-exclusive arguments alter the functionality of the utility so that only certain other arguments are valid in combination with one of the mutually-exclusive arguments. Only one of the mutually-exclusive arguments is allowed for invocation of the utility. Unless otherwise stated in an accompanying OPTIONS section, the relationships between arguments depicted in the SYNOPSIS sections are mandatory requirements placed on conforming applications. The use of conflicting mutually-exclusive arguments produces undefined results, unless a utility description specifies otherwise. When an option is shown without the '[' and ']' brackets, it means that option is required for that version of the SYNOPSIS. However, it is not required to be the first argument, as shown in the example above, unless otherwise stated.

9. Ellipses ("...") are used to denote that one or more occurrences of an operand are allowed. When an option or an operand followed by ellipses is enclosed in brackets, zero or more options or operands can be specified. The form:

```
utility_name [-g option_argument] ... [operand...]
```

indicates that multiple occurrences of the option and its option-argument preceding the ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See also Guideline 11 in [Section 12.2](#).)

The form:

```
utility_name -f option_argument [-f option_argument] ... [operand...]
```

indicates that the -f option is required to appear at least once and may appear multiple times.

10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities volume of POSIX.1-2008, the indented lines following the initial line are continuation lines. An actual use of the command would appear on a single logical line.

12.2 Utility Syntax Guidelines

The following guidelines are established for the naming of utilities and for the specification of options, option-arguments, and operands. The *getopt()* function in the System Interfaces volume of POSIX.1-2008 assists utilities in handling options and operands that conform to these guidelines.

Operands and option-arguments can contain characters not specified in the portable character set.

The guidelines are intended to provide guidance to the authors of future utilities, such as those written specific to a local system or that are components of a larger application. Some of the standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections describe the deviations.

- 7112 **Guideline 1:** Utility names should be between two and nine characters, inclusive.
- 7113 **Guideline 2:** Utility names should include lowercase letters (the **lower** character
- 7114 classification) and digits only from the portable character set.
- 7115 **Guideline 3:** Each option name should be a single alphanumeric character (the **alnum**
- 7116 character classification) from the portable character set. The **-W** (capital-W)
- 7117 option shall be reserved for vendor options.
- 7118 Multi-digit options should not be allowed.
- 7119 **Guideline 4:** All options should be preceded by the **'-'** delimiter character.
- 7120 **Guideline 5:** Options without option-arguments should be accepted when grouped behind
- 7121 one **'-'** delimiter.
- 7122 **Guideline 6:** Each option and option-argument should be a separate argument, except as
- 7123 noted in [Section 12.1](#) (on page 213), item (2).
- 7124 **Guideline 7:** Option-arguments should not be optional.
- 7125 **Guideline 8:** When multiple option-arguments are specified to follow a single option, they
- 7126 should be presented as a single argument, using **<comma>** characters within
- 7127 that argument or **<blank>** characters within that argument to separate them.
- 7128 **Guideline 9:** All options should precede operands on the command line.
- 7129 **Guideline 10:** The first **--** argument that is not an option-argument should be accepted as a
- 7130 delimiter indicating the end of options. Any following arguments should be
- 7131 treated as operands, even if they begin with the **'-'** character.
- 7132 **Guideline 11:** The order of different options relative to one another should not matter, unless
- 7133 the options are documented as mutually-exclusive and such an option is
- 7134 documented to override any incompatible options preceding it. If an option
- 7135 that has option-arguments is repeated, the option and option-argument
- 7136 combinations should be interpreted in the order specified on the command
- 7137 line.
- 7138 **Guideline 12:** The order of operands may matter and position-related interpretations should
- 7139 be determined on a utility-specific basis.
- 7140 **Guideline 13:** For utilities that use operands to represent files to be opened for either reading
- 7141 or writing, the **'-'** operand should be used to mean only standard input (or
- 7142 standard output when it is clear from context that an output file is being
- 7143 specified) or a file named **-**.
- 7144 **Guideline 14:** If an argument can be identified according to Guidelines 3 through 10 as an
- 7145 option, or as a group of options without option-arguments behind one **'-'**
- 7146 delimiter, then it should be treated as such.
- 7147 The utilities in the Shell and Utilities volume of POSIX.1-2008 that claim conformance to these
- 7148 guidelines shall conform completely to these guidelines as if these guidelines contained the term
- 7149 "shall" instead of "should". On some implementations, the utilities accept usage in violation of
- 7150 these guidelines for backwards-compatibility as well as accepting the required form.
- 7151 Where a utility described in the Shell and Utilities volume of POSIX.1-2008 as conforming to
- 7152 these guidelines is required to accept, or not to accept, the operand **'-'** to mean standard input
- 7153 or output, this usage is explained in the OPERANDS section. Otherwise, if such a utility uses
- 7154 operands to represent files, it is implementation-defined whether the operand **'-'** stands for
- 7155 standard input (or standard output), or for a file named **-**.

*Utility Conventions**Utility Syntax Guidelines*

7156 It is recommended that all future utilities and applications use these guidelines to enhance user
7157 portability. The fact that some historical utilities could not be changed (to avoid breaking
7158 existing applications) should not deter this future goal.

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

(blank page)

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



This chapter describes the contents of headers.

Headers contain function prototypes, the definition of symbolic constants, common structures, preprocessor macros, and defined types. Each function in the System Interfaces volume of POSIX.1-2008 specifies the headers that an application shall include in order to use that function. In most cases, only one header is required. These headers are present on an application development system; they need not be present on the target execution system.

Format of Entries

The entries in this chapter are based on a common format as follows. The only sections relating to conformance are the SYNOPSIS and DESCRIPTION.

NAME

This section gives the name or names of the entry and briefly states its purpose.

SYNOPSIS

This section summarizes the use of the entry being described.

DESCRIPTION

This section describes the functionality of the header.

APPLICATION USAGE

This section is informative. This section gives warnings and advice to application developers about the entry. In the event of conflict between warnings and advice and a normative part of this volume of POSIX.1-2008, the normative material is to be taken as correct.

RATIONALE

This section is informative. This section contains historical information concerning the contents of this volume of POSIX.1-2008 and why features were included or discarded by the standard developers.

FUTURE DIRECTIONS

This section is informative. This section provides comments which should be used as a guide to current thinking; there is not necessarily a commitment to adopt these future directions.

SEE ALSO

This section is informative. This section gives references to related information.

CHANGE HISTORY

This section is informative. This section shows the derivation of the entry and any significant changes that have been made to it.

<aio.h>*Headers*7194 **NAME**

7195 aio.h — asynchronous input and output

7196 **SYNOPSIS**

7197 #include <aio.h>

7198 **DESCRIPTION**7199 The **<aio.h>** header shall define the **aio_cb** structure, which shall include at least the following
7200 members:

7201	int	aio_fildes	File descriptor.
7202	off_t	aio_offset	File offset.
7203	volatile void	*aio_buf	Location of buffer.
7204	size_t	aio_nbytes	Length of transfer.
7205	int	aio_reqprio	Request priority offset.
7206	struct sigevent	aio_sigevent	Signal number and value.
7207	int	aio_lio_opcode	Operation to be performed.

7208 The **<aio.h>** header shall define the **off_t**, **pthread_attr_t**, **size_t**, and **ssize_t** types as described
7209 in **<sys/types.h>**.7210 The **<aio.h>** header shall define the **struct timespec** structure as described in **<time.h>**.7211 The tag **sigevent** shall be declared as naming an incomplete structure type, the contents of which
7212 are described in the **<signal.h>** header.7213 The **<aio.h>** header shall define the following symbolic constants:7214 **AIO_ALLDONE** A return value indicating that none of the requested operations could be
7215 canceled since they are already complete.7216 **AIO_CANCELED** A return value indicating that all requested operations have been
7217 canceled.7218 **AIO_NOTCANCELED**7219 A return value indicating that some of the requested operations could not
7220 be canceled since they are in progress.7221 **LIO_NOP** A *lio_listio()* element operation option indicating that no transfer is
7222 requested.7223 **LIO_NOWAIT** A *lio_listio()* synchronization operation indicating that the calling thread
7224 is to continue execution while the *lio_listio()* operation is being
7225 performed, and no notification is given when the operation is complete.7226 **LIO_READ** A *lio_listio()* element operation option requesting a read.7227 **LIO_WAIT** A *lio_listio()* synchronization operation indicating that the calling thread
7228 is to suspend until the *lio_listio()* operation is complete.7229 **LIO_WRITE** A *lio_listio()* element operation option requesting a write.7230 The following shall be declared as functions and may also be defined as macros. Function
7231 prototypes shall be provided.

```

7232 int      aio_cancel(int, struct aiocb *);
7233 int      aio_error(const struct aiocb *);
7234 int      aio_fsync(int, struct aiocb *);
7235 int      aio_read(struct aiocb *);
7236 ssize_t  aio_return(struct aiocb *);
7237 int      aio_suspend(const struct aiocb *const [], int,
```


Headers

<aio.h>

```

7238         const struct timespec *);
7239     int      aio_write(struct aiocb *);
7240     int      lio_listio(int, struct aiocb *restrict const [restrict], int,
7241         struct sigevent *restrict);

```

7242 Inclusion of the <aio.h> header may make visible symbols defined in the headers <fcntl.h>,
 7243 <signal.h>, and <time.h>.

7244 APPLICATION USAGE

7245 None.

7246 RATIONALE

7247 None.

7248 FUTURE DIRECTIONS

7249 None.

7250 SEE ALSO

7251 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>

7252 XSH *aio_cancel()*, *aio_error()*, *aio_fsync()*, *aio_read()*, *aio_return()*, *aio_suspend()*, *aio_write()*,
 7253 *fsync()*, *lio_listio()*, *lseek()*, *read()*, *write()*

7254 CHANGE HISTORY

7255 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

7256 Issue 6

7257 The <aio.h> header is marked as part of the Asynchronous Input and Output option.

7258 The description of the constants is expanded.

7259 The **restrict** keyword is added to the prototype for *lio_listio()*.

7260 Issue 7

7261 The <aio.h> header is moved from the Asynchronous Input and Output option to the Base.

7262 This reference page is clarified with respect to macros and symbolic constants, and type and
 7263 structure declarations are added.

<arpa/inet.h>

Headers

7264 **NAME**

7265 arpa/inet.h — definitions for internet operations

7266 **SYNOPSIS**

7267 #include <arpa/inet.h>

7268 **DESCRIPTION**7269 The **<arpa/inet.h>** header shall define the **in_port_t** and **in_addr_t** types as described in
7270 **<netinet/in.h>**.7271 The **<arpa/inet.h>** header shall define the **in_addr** structure as described in **<netinet/in.h>**.7272 IP6 The **<arpa/inet.h>** header shall define the INET_ADDRSTRLEN and INET6_ADDRSTRLEN
7273 macros as described in **<netinet/in.h>**.7274 The following shall be declared as functions, or defined as macros, or both. If functions are
7275 declared, function prototypes shall be provided.

7276 uint32_t htonl(uint32_t);

7277 uint16_t htons(uint16_t);

7278 uint32_t ntohl(uint32_t);

7279 uint16_t ntohs(uint16_t);

7280 The **<arpa/inet.h>** header shall define the **uint32_t** and **uint16_t** types as described in
7281 **<inttypes.h>**.7282 The following shall be declared as functions and may also be defined as macros. Function
7283 prototypes shall be provided.

7284 in_addr_t inet_addr(const char*);

7285 char *inet_ntoa(struct in_addr);

7286 const char *inet_ntop(int, const void *restrict, char *restrict,
7287 socklen_t);

7288 int inet_pton(int, const char *restrict, void *restrict);

7289 Inclusion of the **<arpa/inet.h>** header may also make visible all symbols from **<netinet/in.h>**
7290 and **<inttypes.h>**.7291 **APPLICATION USAGE**

7292 None.

7293 **RATIONALE**

7294 None.

7295 **FUTURE DIRECTIONS**

7296 None.

7297 **SEE ALSO**7298 **<inttypes.h>**, **<netinet/in.h>**7299 XSH *htonl()*, *inet_addr()*, *inet_ntop()*7300 **CHANGE HISTORY**

7301 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7302 The **restrict** keyword is added to the prototypes for *inet_ntop()* and *inet_pton()*.7303 **Issue 7**

7304 SD5-XBD-ERN-6 is applied.

Headers

<assert.h>

NAME

assert.h — verify program assertion

SYNOPSIS

```
#include <assert.h>
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The <assert.h> header shall define the *assert()* macro. It refers to the macro NDEBUG which is not defined in the header. If NDEBUG is defined as a macro name before the inclusion of this header, the *assert()* macro shall be defined simply as:

```
#define assert(ignore) ((void) 0)
```

Otherwise, the macro behaves as described in *assert()*.

The *assert()* macro shall be redefined according to the current state of NDEBUG each time <assert.h> is included.

The *assert()* macro shall be implemented as a macro, not as a function. If the macro definition is suppressed in order to access an actual function, the behavior is undefined.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *assert()*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The definition of the *assert()* macro is changed for alignment with the ISO/IEC 9899:1999 standard.

<complex.h>

Headers

7335 NAME

7336 complex.h — complex arithmetic

7337 SYNOPSIS

7338 #include <complex.h>

7339 DESCRIPTION

7340 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 7341 conflict between the requirements described here and the ISO C standard is unintentional. This
 7342 volume of POSIX.1-2008 defers to the ISO C standard.

7343 The <complex.h> header shall define the following macros:

7344 complex Expands to **_Complex**.

7345 **_Complex_I** Expands to a constant expression of type **const float _Complex**, with the value
 7346 of the imaginary unit (that is, a number i such that $i^2 = -1$).

7347 imaginary Expands to **_Imaginary**.

7348 **_Imaginary_I** Expands to a constant expression of type **const float _Imaginary** with the
 7349 value of the imaginary unit.

7350 **I** Expands to either **_Imaginary_I** or **_Complex_I**. If **_Imaginary_I** is not defined,
 7351 **I** expands to **_Complex_I**.

7352 The macros **imaginary** and **_Imaginary_I** shall be defined if and only if the implementation
 7353 supports imaginary types.

7354 An application may undefine and then, perhaps, redefine the **complex**, **imaginary**, and **I** macros.

7355 The following shall be declared as functions and may also be defined as macros. Function
 7356 prototypes shall be provided.

7357	double	cabs(double complex);
7358	float	cabsf(float complex);
7359	long double	cabsl(long double complex);
7360	double complex	cacos(double complex);
7361	float complex	cacosf(float complex);
7362	double complex	cacosh(double complex);
7363	float complex	cacoshf(float complex);
7364	long double complex	cacoshl(long double complex);
7365	long double complex	cacosl(long double complex);
7366	double	carg(double complex);
7367	float	cargf(float complex);
7368	long double	cargl(long double complex);
7369	double complex	casin(double complex);
7370	float complex	casinf(float complex);
7371	double complex	casinh(double complex);
7372	float complex	casinhf(float complex);
7373	long double complex	casinhl(long double complex);
7374	long double complex	casinl(long double complex);
7375	double complex	catan(double complex);
7376	float complex	catanf(float complex);
7377	double complex	catanh(double complex);
7378	float complex	catanhf(float complex);
7379	long double complex	catanhl(long double complex);
7380	long double complex	catanl(long double complex);

Headers

<complex.h>

7381	double complex	ccos(double complex);
7382	float complex	ccosf(float complex);
7383	double complex	ccosh(double complex);
7384	float complex	ccoshf(float complex);
7385	long double complex	ccoshl(long double complex);
7386	long double complex	ccosl(long double complex);
7387	double complex	cexp(double complex);
7388	float complex	cexpf(float complex);
7389	long double complex	cexpl(long double complex);
7390	double	cimag(double complex);
7391	float	cimagf(float complex);
7392	long double	cimagl(long double complex);
7393	double complex	clog(double complex);
7394	float complex	clogf(float complex);
7395	long double complex	clogl(long double complex);
7396	double complex	conj(double complex);
7397	float complex	conjf(float complex);
7398	long double complex	conjl(long double complex);
7399	double complex	cpow(double complex, double complex);
7400	float complex	cpowf(float complex, float complex);
7401	long double complex	cpowl(long double complex, long double complex);
7402	double complex	cproj(double complex);
7403	float complex	cprojf(float complex);
7404	long double complex	cprojl(long double complex);
7405	double	creal(double complex);
7406	float	crealf(float complex);
7407	long double	creall(long double complex);
7408	double complex	csin(double complex);
7409	float complex	csinf(float complex);
7410	double complex	csinh(double complex);
7411	float complex	csinhf(float complex);
7412	long double complex	csinhl(long double complex);
7413	long double complex	csinl(long double complex);
7414	double complex	csqrt(double complex);
7415	float complex	csqrtf(float complex);
7416	long double complex	csqrtl(long double complex);
7417	double complex	ctan(double complex);
7418	float complex	ctanf(float complex);
7419	double complex	ctanh(double complex);
7420	float complex	ctanhf(float complex);
7421	long double complex	ctanhl(long double complex);
7422	long double complex	ctanl(long double complex);

<complex.h>

Headers

APPLICATION USAGE

Values are interpreted as radians, not degrees.

RATIONALE

The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the identifier *i* for other purposes. The application can use a different identifier, say *j*, for the imaginary unit by following the inclusion of the <complex.h> header with:

```
#undef I
#define j _Imaginary_I
```

An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a sufficiently convenient and more generally useful notation for imaginary terms. The corresponding real type for the imaginary unit is **float**, so that use of *I* for algorithmic or notational convenience will not result in widening types.

On systems with imaginary types, the application has the ability to control whether use of the macro *I* introduces an imaginary type, by explicitly defining *I* to be *_Imaginary_I* or *_Complex_I*. Disallowing imaginary types is useful for some applications intended to run on implementations without support for such types.

The macro *_Imaginary_I* provides a test for whether imaginary types are supported.

The *cis()* function ($\cos(x) + I\sin(x)$) was considered but rejected because its implementation is easy and straightforward, even though some implementations could compute sine and cosine more efficiently in tandem.

FUTURE DIRECTIONS

The following function names and the same names suffixed with *f* or *l* are reserved for future use, and may be added to the declarations in the <complex.h> header.

<i>cerf()</i>	<i>cexpm1()</i>	<i>clog2()</i>
<i>cerfc()</i>	<i>clog10()</i>	<i>clgamma()</i>
<i>cexp2()</i>	<i>clog1p()</i>	<i>ctgamma()</i>

SEE ALSO

XSH *cabs()*, *cacos()*, *cacosh()*, *carg()*, *casin()*, *casinh()*, *catan()*, *catanh()*, *ccos()*, *ccosh()*, *cexp()*, *cimag()*, *clog()*, *conj()*, *cpow()*, *cproj()*, *creal()*, *csin()*, *csinh()*, *csqrt()*, *ctan()*, *ctanh()*

CHANGE HISTORY

First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

Headers

<cpio.h>

NAME

cpio.h — cpio archive values

SYNOPSIS

#include <cpio.h>

DESCRIPTION

The <cpio.h> header shall define the symbolic constants needed by the *c_mode* field of the *cpio* archive format, with the names and values given in the following table:

Name	Description	Value (Octal)
C_IRUSR	Read by owner.	0000400
C_IWUSR	Write by owner.	0000200
C_IXUSR	Execute by owner.	0000100
C_IRGRP	Read by group.	0000040
C_IWGRP	Write by group.	0000020
C_IXGRP	Execute by group.	0000010
C_IROTH	Read by others.	0000004
C_IWOTH	Write by others.	0000002
C_IXOTH	Execute by others.	0000001
C_ISUID	Set user ID.	0004000
C_ISGID	Set group ID.	0002000
C_ISVTX	On directories, restricted deletion flag.	0001000
C_ISDIR	Directory.	0040000
C_ISFIFO	FIFO.	0010000
C_ISREG	Regular file.	0100000
C_ISBLK	Block special.	0060000
C_ISCHR	Character special.	0020000
C_ISCTG	Reserved.	0110000
C_ISLNK	Symbolic link.	0120000
C_ISSOCK	Socket.	0140000

The <cpio.h> header shall define the following symbolic constant as a string:

MAGIC "070707"

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XCU *pax*

CHANGE HISTORY

First released in the Headers Interface, Issue 3 specification. Derived from the POSIX.1-1988 standard.

Issue 6

The SEE ALSO is updated to refer to *pax*.

<cpio.h>*Headers***7497 Issue 7**

7498 The **<cpio.h>** header is moved from the XSI option to the Base.

7499 This reference page is clarified with respect to macros and symbolic constants.

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

Headers

<ctype.h>

NAME

ctype.h — character types

SYNOPSIS

```
#include <ctype.h>
```

DESCRIPTION

Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to enable the visibility of these symbols in this header.

The <ctype.h> header shall define the **locale_t** type as described in <locale.h>, representing a locale object.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
int    isalnum(int);
int    isalnum_l(int, locale_t);
int    isalpha(int);
int    isalpha_l(int, locale_t);
int    isascii(int);
int    isblank(int);
int    isblank_l(int, locale_t);
int    iscntrl(int);
int    iscntrl_l(int, locale_t);
int    isdigit(int);
int    isdigit_l(int, locale_t);
int    isgraph(int);
int    isgraph_l(int, locale_t);
int    islower(int);
int    islower_l(int, locale_t);
int    isprint(int);
int    isprint_l(int, locale_t);
int    ispunct(int);
int    ispunct_l(int, locale_t);
int    isspace(int);
int    isspace_l(int, locale_t);
int    isupper(int);
int    isupper_l(int, locale_t);
int    isxdigit(int);
int    isxdigit_l(int, locale_t);
int    toascii(int);
int    tolower(int);
int    tolower_l(int, locale_t);
int    toupper(int);
int    toupper_l(int, locale_t);
```

The <ctype.h> header shall define the following as macros:

```
int    _toupper(int);
int    _tolower(int);
```

<ctype.h>

Headers

7545 **APPLICATION USAGE**

7546 None.

7547 **RATIONALE**

7548 None.

7549 **FUTURE DIRECTIONS**

7550 None.

7551 **SEE ALSO**7552 **<locale.h>**

7553 XSH Section 2.2 (on page 468), *isalnum()*, *isalpha()*, *isascii()*, *isblank()*, *iscntrl()*, *isdigit()*,
 7554 *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *mblen()*, *mbstowcs()*,
 7555 *mbtowc()*, *setlocale()*, *toascii()*, *tolower()*, *_tolower()*, *toupper()*, *_toupper()*, *wcstombs()*, *wctomb()*

7556 **CHANGE HISTORY**

7557 First released in Issue 1. Derived from Issue 1 of the SVID.

7558 **Issue 6**

7559 Extensions beyond the ISO C standard are marked.

7560 **Issue 7**

7561 SD5-XBD-ERN-6 is applied, updating the wording regarding the function declarations for
 7562 consistency.

7563 The *_l() functions are added from The Open Group Technical Standard, 2006, Extended API Set
 7564 Part 4.

Headers

<dirent.h>

NAME

dirent.h — format of directory entries

SYNOPSIS

```
#include <dirent.h>
```

DESCRIPTION

The internal format of directories is unspecified.

The <dirent.h> header shall define the following type:

DIR A type representing a directory stream. The **DIR** type may be an incomplete type.

It shall also define the structure **dirent** which shall include the following members:

XSI	ino_t d_ino	File serial number.
	char d_name[]	Name of entry.

XSI The <dirent.h> header shall define the **ino_t** type as described in <sys/types.h>.

The character array *d_name* is of unspecified size, but the number of bytes preceding the terminating null byte shall not exceed {NAME_MAX}.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

int	alphasort(const struct dirent **, const struct dirent **);
int	closedir(DIR *);
int	dirfd(DIR *);
DIR	*fdopendir(int);
DIR	*opendir(const char *);
struct dirent	*readdir(DIR *);
int	readdir_r(DIR *restrict, struct dirent *restrict,
	struct dirent **restrict);
void	rewinddir(DIR *);
int	scandir(const char *, struct dirent **,
	int (*)(const struct dirent *),
	int (*)(const struct dirent **,
	const struct dirent **));
XSI	void seekdir(DIR *, long);
long	telldir(DIR *);

APPLICATION USAGE

None.

RATIONALE

Information similar to that in the <dirent.h> header is contained in a file <sys/dir.h> in 4.2 BSD and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of POSIX.1-2008 is **struct direct**. The filename was changed because the name <sys/dir.h> was also used in earlier implementations to refer to definitions related to the older access method; this produced name conflicts. The name of the structure was changed because this volume of POSIX.1-2008 does not completely define what is in the structure, so it could be different on some implementations from **struct direct**.

The name of an array of **char** of an unspecified size should not be used as an lvalue. Use of:

```
sizeof(d_name)
```

is incorrect; use:

<dirent.h>

Headers

7609 `strlen(d_name)`

7610 instead.

7611 The array of **char** *d_name* is not a fixed size. Implementations may need to declare **struct dirent**
 7612 with an array size for *d_name* of 1, but the actual number of characters provided matches (or
 7613 only slightly exceeds) the length of the filename.

7614 FUTURE DIRECTIONS

7615 None.

7616 SEE ALSO

7617 [`<sys/types.h>`](#)

7618 XSH *alphasort()*, *closedir()*, *dirfd()*, *fdopendir()*, *readdir()*, *rewinddir()*, *seekdir()*, *telldir()*

7619 CHANGE HISTORY

7620 First released in Issue 2.

7621 Issue 5

7622 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

7623 Issue 6

7624 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r()*.

7625 The **restrict** keyword is added to the prototype for *readdir_r()*.

7626 Issue 7

7627 The *alphasort()*, *dirfd()*, and *scandir()* functions are added from The Open Group Technical
 7628 Standard, 2006, Extended API Set Part 1.

7629 The *fopendir()* function is added from The Open Group Technical Standard, 2006, Extended API
 7630 Set Part 2.

7631 Austin Group Interpretation 1003.1-2001 #110 is applied, clarifying the definition of the **DIR**
 7632 type.

Headers

<dlfcn.h>

NAME

dlfcn.h — dynamic linking

SYNOPSIS

```
#include <dlfcn.h>
```

DESCRIPTION

The <dlfcn.h> header shall define at least the following symbolic constants for use in the construction of a *dlopen()* *mode* argument:

RTLD_LAZY	Relocations are performed at an implementation-defined time.
RTLD_NOW	Relocations are performed when the object is loaded.
RTLD_GLOBAL	All symbols are available for relocation processing of other modules.
RTLD_LOCAL	All symbols are not made available for relocation processing by other modules.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int    dlclose(void *);
char   *dLError(void);
void   *dlopen(const char *, int);
void   *dlsym(void *restrict, const char *restrict);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *dlclose()*, *dLError()*, *dlopen()*, *dlsym()*

CHANGE HISTORY

First released in Issue 5.

Issue 6

The **restrict** keyword is added to the prototype for *dlsym()*.

Issue 7

The <dlfcn.h> header is moved from the XSI option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

<errno.h>*Headers*7666 **NAME**7667 `errno.h` — system error numbers7668 **SYNOPSIS**7669 `#include <errno.h>`7670 **DESCRIPTION**

7671 CX Some of the functionality described on this reference page extends the ISO C standard. Any
 7672 conflict between the requirements described here and the ISO C standard is unintentional. This
 7673 volume of POSIX.1-2008 defers to the ISO C standard.

7674 The ISO C standard only requires the symbols [EDOM], [EILSEQ], and [ERANGE] to be defined.

7675 The **<errno.h>** header shall provide a declaration or definition for *errno*. The symbol *errno* shall
 7676 expand to a modifiable lvalue of type **int**. It is unspecified whether *errno* is a macro or an
 7677 identifier declared with external linkage. If a macro definition is suppressed in order to access an
 7678 actual object, or a program defines an identifier with the name *errno*, the behavior is undefined.

7679 The **<errno.h>** header shall define the following macros which shall expand to integer constant
 7680 expressions with type **int**, distinct positive values (except as noted below), and which shall be
 7681 suitable for use in **#if** preprocessing directives:

7682	[E2BIG]	Argument list too long.
7683	[EACCES]	Permission denied.
7684	[EADDRINUSE]	Address in use.
7685	[EADDRNOTAVAIL]	Address not available.
7686	[EAFNOSUPPORT]	Address family not supported.
7687	[EAGAIN]	Resource unavailable, try again (may be the same value as [EWOULDBLOCK]).
7688		
7689	[EALREADY]	Connection already in progress.
7690	[EBADF]	Bad file descriptor.
7691	[EBADMSG]	Bad message.
7692	[EBUSY]	Device or resource busy.
7693	[ECANCELED]	Operation canceled.
7694	[ECHILD]	No child processes.
7695	[ECONNABORTED]	Connection aborted.
7696	[ECONNREFUSED]	Connection refused.
7697	[ECONNRESET]	Connection reset.
7698	[EDEADLK]	Resource deadlock would occur.
7699	[EDESTADDRREQ]	Destination address required.
7700	[EDOM]	Mathematics argument out of domain of function.
7701	[EDQUOT]	Reserved.
7702	[EEXIST]	File exists.

Headers

<errno.h>

7703		[EFAULT]	Bad address.
7704		[EFBIG]	File too large.
7705		[EHOSTUNREACH]	Host is unreachable.
7706		[EIDRM]	Identifier removed.
7707		[EILSEQ]	Illegal byte sequence.
7708		[EINPROGRESS]	Operation in progress.
7709		[EINTR]	Interrupted function.
7710		[EINVAL]	Invalid argument.
7711		[EIO]	I/O error.
7712		[EISCONN]	Socket is connected.
7713		[EISDIR]	Is a directory.
7714		[ELOOP]	Too many levels of symbolic links.
7715		[EMFILE]	File descriptor value too large.
7716		[EMLINK]	Too many links.
7717		[EMSGSIZE]	Message too large.
7718		[EMULTIHOP]	Reserved.
7719		[ENAMETOOLONG]	Filename too long.
7720		[ENETDOWN]	Network is down.
7721		[ENETRESET]	Connection aborted by network.
7722		[ENETUNREACH]	Network unreachable.
7723		[ENFILE]	Too many files open in system.
7724		[ENOBUFS]	No buffer space available.
7725	OB XSR	[ENODATA]	No message is available on the STREAM head read queue.
7726		[ENODEV]	No such device.
7727		[ENOENT]	No such file or directory.
7728		[ENOEXEC]	Executable file format error.
7729		[ENOLCK]	No locks available.
7730		[ENOLINK]	Reserved.
7731		[ENOMEM]	Not enough space.
7732		[ENOMSG]	No message of the desired type.
7733		[ENOPROTOOPT]	Protocol not available.
7734		[ENOSPC]	No space left on device.
7735	OB XSR	[ENOSR]	No STREAM resources.

<errno.h>

Headers

7736	OB XSR	[ENOSTR]	Not a STREAM.
7737		[ENOSYS]	Function not supported.
7738		[ENOTCONN]	The socket is not connected.
7739		[ENOTDIR]	Not a directory.
7740		[ENOTEMPTY]	Directory not empty.
7741		[ENOTRECOVERABLE]	
7742			State not recoverable.
7743		[ENOTSOCK]	Not a socket.
7744		[ENOTSUP]	Not supported (may be the same value as [EOPNOTSUPP]).
7745		[ENOTTY]	Inappropriate I/O control operation.
7746		[ENXIO]	No such device or address.
7747		[EOPNOTSUPP]	Operation not supported on socket (may be the same value as
7748			[ENOTSUP]).
7749		[EOVERFLOW]	Value too large to be stored in data type.
7750		[EOWNERDEAD]	Previous owner died.
7751		[EPERM]	Operation not permitted.
7752		[EPIPE]	Broken pipe.
7753		[EPROTO]	Protocol error.
7754		[EPROTONOSUPPORT]	
7755			Protocol not supported.
7756		[EPROTOTYPE]	Protocol wrong type for socket.
7757		[ERANGE]	Result too large.
7758		[EROFS]	Read-only file system.
7759		[ESPIPE]	Invalid seek.
7760		[ESRCH]	No such process.
7761		[ESTALE]	Reserved.
7762	OB XSR	[ETIME]	Stream <i>ioctl()</i> timeout.
7763		[ETIMEDOUT]	Connection timed out.
7764		[ETXTBSY]	Text file busy.
7765		[EWOULDBLOCK]	Operation would block (may be the same value as [EAGAIN]).
7766		[EXDEV]	Cross-device link.

Headers

<errno.h>

APPLICATION USAGE

Additional error numbers may be defined on conforming systems; see the System Interfaces volume of POSIX.1-2008.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH [Section 2.3](#) (on page 477)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Updated for alignment with the POSIX Realtime Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The majority of the error conditions previously marked as extensions are now mandatory, except for the STREAMS-related error conditions.

Values for *errno* are now required to be distinct positive values rather than non-zero values. This change is for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #050 is applied, allowing [ENOTSUP] and [EOPNOTSUPP] to be the same values.

The [ENOTRECOVERABLE] and [EOWNERDEAD] errors are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Functionality relating to the Threads option is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants.

<fcntl.h>*Headers*7795 **NAME**7796 `fcntl.h` — file control options7797 **SYNOPSIS**7798 `#include <fcntl.h>`7799 **DESCRIPTION**

7800 The **<fcntl.h>** header shall define the following symbolic constants for the *cmd* argument used
 7801 by *fcntl()*. The values shall be unique and shall be suitable for use in **#if** preprocessing
 7802 directives.

7803 **F_DUPFD** Duplicate file descriptor.

7804 **F_DUPFD_CLOEXEC**

7805 Duplicate file descriptor with the close-on-exec flag **FD_CLOEXEC** set.

7806 **F_GETFD** Get file descriptor flags.

7807 **F_SETFD** Set file descriptor flags.

7808 **F_GETFL** Get file status flags and file access modes.

7809 **F_SETFL** Set file status flags.

7810 **F_GETLK** Get record locking information.

7811 **F_SETLK** Set record locking information.

7812 **F_SETLKW** Set record locking information; wait if blocked.

7813 **F_GETOWN** Get process or process group ID to receive SIGURG signals.

7814 **F_SETOWN** Set process or process group ID to receive SIGURG signals.

7815 The **<fcntl.h>** header shall define the following symbolic constant used for the *fcntl()* file
 7816 descriptor flags, which shall be suitable for use in **#if** preprocessing directives.

7817 **FD_CLOEXEC** Close the file descriptor upon execution of an *exec* family function.

7818 The **<fcntl.h>** header shall also define the following symbolic constants for the *l_type* argument
 7819 used for record locking with *fcntl()*. The values shall be unique and shall be suitable for use in
 7820 **#if** preprocessing directives.

7821 **F_RDLCK** Shared or read lock.

7822 **F_UNLCK** Unlock.

7823 **F_WRLCK** Exclusive or write lock.

7824 The **<fcntl.h>** header shall define the values used for *l_whence*, **SEEK_SET**, **SEEK_CUR**, and
 7825 **SEEK_END** as described in **<stdio.h>**.

7826 The **<fcntl.h>** header shall define the following symbolic constants as file creation flags for use
 7827 in the *oflag* value to *open()* and *openat()*. The values shall be bitwise-distinct and shall be
 7828 suitable for use in **#if** preprocessing directives.

7829 **O_CREAT** Create file if it does not exist.

7830 **O_EXCL** Exclusive use flag.

7831 **O_NOCTTY** Do not assign controlling terminal.

7832 **O_TRUNC** Truncate flag.

Headers

<fcntl.h>

7833	O_TTY_INIT	Set the termios structure terminal parameters to a state that provides conforming behavior; see Section 11.2 (on page 205).
7834		
7835		The O_TTY_INIT flag can have the value zero and in this case it need not be bitwise-distinct from the other flags.
7836		
7837		The <fcntl.h> header shall define the following symbolic constants for use as file status flags for <i>open()</i> , <i>openat()</i> , and <i>fcntl()</i> . The values shall be suitable for use in #if preprocessing directives.
7838		
7839	O_APPEND	Set append mode.
7840	SIO O_DSYNC	Write according to synchronized I/O data integrity completion.
7841	O_NONBLOCK	Non-blocking mode.
7842	SIO O_RSYNC	Synchronized read I/O operations.
7843	O_SYNC	Write according to synchronized I/O file integrity completion.
7844		The <fcntl.h> header shall define the following symbolic constant for use as the mask for file access modes. The value shall be suitable for use in #if preprocessing directives.
7845		
7846	O_ACCMODE	Mask for file access modes.
7847		The <fcntl.h> header shall define the following symbolic constants for use as the file access modes for <i>open()</i> , <i>openat()</i> , and <i>fcntl()</i> . The values shall be suitable for use in #if preprocessing directives.
7848		
7849		
7850	O_EXEC	Open for execute only (non-directory files). The result is unspecified if this flag is applied to a directory.
7851		
7852	O_RDONLY	Open for reading only.
7853	O_RDWR	Open for reading and writing.
7854	O_SEARCH	Open directory for search only. The result is unspecified if this flag is applied to a non-directory file.
7855		
7856	O_WRONLY	Open for writing only.
7857		The <fcntl.h> header shall define the symbolic constants for file modes for use as values of mode_t as described in <sys/stat.h>.
7858		
7859		The <fcntl.h> header shall define the following symbolic constant as a special value used in place of a file descriptor for the <i>*at()</i> functions which take a directory file descriptor as a parameter:
7860		
7861		
7862	AT_FDCWD	Use the current working directory to determine the target of relative file paths.
7863		The <fcntl.h> header shall define the following symbolic constant as a value for the <i>flag</i> used by <i>faccessat()</i> :
7864		
7865	AT_EACCESS	Check access using effective user and group ID.
7866		The <fcntl.h> header shall define the following symbolic constant as a value for the <i>flag</i> used by <i>fstatat()</i> , <i>fchmodat()</i> , <i>fchownat()</i> , and <i>utimensat()</i> :
7867		
7868	AT_SYMLINK_NOFOLLOW	
7869		Do not follow symbolic links.
7870		The <fcntl.h> header shall define the following symbolic constant as a value for the flag used by <i>linkat()</i> :
7871		

<fcntl.h>

Headers

7872 AT_SYMLINK_FOLLOW
7873 Follow symbolic link.

7874 The <fcntl.h> header shall define the following symbolic constants as values for the flag used by
7875 *open()* and *openat()*:

7876 O_CLOEXEC The FD_CLOEXEC flag associated with the new descriptor shall be set to close
7877 the file descriptor upon execution of an *exec* family function.

7878 O_DIRECTORY Fail if not a directory.

7879 O_NOFOLLOW Do not follow symbolic links.

7880 The <fcntl.h> header shall define the following symbolic constant as a value for the flag used by
7881 *unlinkat()*:

7882 AT_REMOVEDIR
7883 Remove directory instead of file.

7884 ADV The <fcntl.h> header shall define the following symbolic constants for the *advice* argument used
7885 by *posix_fadvise()*:

7886 POSIX_FADV_DONTNEED
7887 The application expects that it will not access the specified data in the near future.

7888 POSIX_FADV_NOREUSE
7889 The application expects to access the specified data once and then not reuse it thereafter.

7890 POSIX_FADV_NORMAL
7891 The application has no advice to give on its behavior with respect to the specified data. It is
7892 the default characteristic if no advice is given for an open file.

7893 POSIX_FADV_RANDOM
7894 The application expects to access the specified data in a random order.

7895 POSIX_FADV_SEQUENTIAL
7896 The application expects to access the specified data sequentially from lower offsets to higher
7897 offsets.

7898 POSIX_FADV_WILLNEED
7899 The application expects to access the specified data in the near future.

7900 The <fcntl.h> header shall define the **flock** structure describing a file lock. It shall include the
7901 following members:

7902 short l_type Type of lock; F_RDLCK, F_WRLCK, F_UNLCK.
7903 short l_whence Flag for starting offset.
7904 off_t l_start Relative offset in bytes.
7905 off_t l_len Size; if 0 then until EOF.
7906 pid_t l_pid Process ID of the process holding the lock; returned with F_GETLK.

7907 The <fcntl.h> header shall define the **mode_t**, **off_t**, and **pid_t** types as described in
7908 <sys/types.h>.

7909 The following shall be declared as functions and may also be defined as macros. Function
7910 prototypes shall be provided.

7911 int creat(const char *, mode_t);
7912 int fcntl(int, int, ...);
7913 int open(const char *, int, ...);

Headers

<fcntl.h>

```

7914      int  openat(int, const char *, int, ...);
7915  ADV    int  posix_fadvise(int, off_t, off_t, int);
7916      int  posix_fallocate(int, off_t, off_t);

```

7917 Inclusion of the <fcntl.h> header may also make visible all symbols from <sys/stat.h> and
 7918 <unistd.h>.

7919 APPLICATION USAGE

7920 Although no existing implementation defines AT_SYMLINK_FOLLOW and
 7921 AT_SYMLINK_NOFOLLOW as the same numeric value, POSIX.1-2008 does not prohibit that as
 7922 the two constants are not used with the same interfaces.

7923 RATIONALE

7924 While many of the symbolic constants introduced in the <fcntl.h> header do not strictly need to
 7925 be used in #if preprocessor directives, widespread historic practice has defined them as macros
 7926 that are usable in such constructs, and examination of existing applications has shown that they
 7927 are occasionally used in such a way. Therefore it was decided to retain this requirement on an
 7928 implementation in POSIX.1-2008.

7929 FUTURE DIRECTIONS

7930 None.

7931 SEE ALSO

7932 <stdio.h>, <sys/stat.h>, <sys/types.h>, <unistd.h>

7933 XSH *creat()*, *exec*, *fcntl()*, *futimens()*, *open()*, *posix_fadvise()*, *posix_fallocate()*, *posix_madvise()*

7934 CHANGE HISTORY

7935 First released in Issue 1. Derived from Issue 1 of the SVID.

7936 Issue 5

7937 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

7938 Issue 6

7939 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 7940 • O_DSYNC and O_RSYNC are marked as part of the Synchronized Input and Output
 7941 option.

7942 The following new requirements on POSIX implementations derive from alignment with the
 7943 Single UNIX Specification:

- 7944 • The definition of the **mode_t**, **off_t**, and **pid_t** types is mandated.

7945 The F_GETOWN and F_SETOWN values are added for sockets.

7946 The *posix_fadvise()*, *posix_fallocate()*, and *posix_madvise()* functions are added for alignment with
 7947 IEEE Std 1003.1d-1999.

7948 IEEE PASC Interpretation 1003.1 #102 is applied, moving the prototype for *posix_madvise()* to
 7949 <sys/mman.h>.

7950 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/18 is applied, updating the prototypes for
 7951 *posix_fadvise()* and *posix_fallocate()* to be large file-aware, using **off_t** instead of **size_t**.

7952 Issue 7

7953 Austin Group Interpretation 1003.1-2001 #144 is applied, adding the O_TTY_INIT flag.

7954 Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the
 7955 FD_CLOEXEC flag atomically at *open()*, and adding the F_DUPFD_CLOEXEC flag.

<fcntl.h>

Headers

- 7956 The *openat()* function is added from The Open Group Technical Standard, 2006, Extended API
7957 Set Part 2.
- 7958 Additional flags are added to support *faccessat()*, *fchmodat()*, *fchownat()*, *fstatat()*, *linkat()*,
7959 *open()*, *openat()*, and *unlinkat()*.
- 7960 This reference page is clarified with respect to macros and symbolic constants.
- 7961 Changes are made related to support for finegrained timestamps.
- 7962 Changes are made to allow a directory to be opened for searching.

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

Headers

<fenv.h>

NAME

fenv.h — floating-point environment

SYNOPSIS

```
#include <fenv.h>
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The <fenv.h> header shall define the following data types through **typedef**:

fenv_t Represents the entire floating-point environment. The floating-point environment refers collectively to any floating-point status flags and control modes supported by the implementation.

fexcept_t Represents the floating-point status flags collectively, including any status the implementation associates with the flags. A floating-point status flag is a system variable whose value is set (but never cleared) when a floating-point exception is raised, which occurs as a side-effect of exceptional floating-point arithmetic to provide auxiliary information. A floating-point control mode is a system variable whose value may be set by the user to affect the subsequent behavior of floating-point arithmetic.

The <fenv.h> header shall define each of the following macros if and only if the implementation supports the floating-point exception by means of the floating-point functions *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, and *fetestexcept()*. The defined macros shall expand to integer constant expressions with values that are bitwise-distinct.

```
FE_DIVBYZERO
FE_INEXACT
FE_INVALID
FE_OVERFLOW
FE_UNDERFLOW
```

If the implementation supports the IEC 60559 Floating-Point option, all five macros shall be defined. Additional implementation-defined floating-point exceptions with macros beginning with *FE_* and an uppercase letter may also be specified by the implementation.

The <fenv.h> header shall define the macro *FE_ALL_EXCEPT* as the bitwise-inclusive OR of all floating-point exception macros defined by the implementation, if any. If no such macros are defined, then the macro *FE_ALL_EXCEPT* shall be defined as zero.

The <fenv.h> header shall define each of the following macros if and only if the implementation supports getting and setting the represented rounding direction by means of the *fegetround()* and *fesetround()* functions. The defined macros shall expand to integer constant expressions whose values are distinct non-negative values.

```
FE_DOWNWARD
FE_TONEAREST
FE_TOWARDZERO
FE_UPWARD
```

If the implementation supports the IEC 60559 Floating-Point option, all four macros shall be defined. Additional implementation-defined rounding directions with macros beginning with *FE_* and an uppercase letter may also be specified by the implementation.

<fenv.h>*Headers*

The **<fenv.h>** header shall define the following macro, which represents the default floating-point environment (that is, the one installed at program startup) and has type pointer to const-qualified **fenv_t**. It can be used as an argument to the functions within the **<fenv.h>** header that manage the floating-point environment.

FE_DFL_ENV

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int    feclearexcept(int);
int    fegetenv(fenv_t *);
int    fegetexceptflag(fexcept_t *, int);
int    fegetround(void);
int    feholdexcept(fenv_t *);
int    feraiseexcept(int);
int    fesetenv(const fenv_t *);
int    fesetexceptflag(const fexcept_t *, int);
int    fesetround(int);
int    fetestexcept(int);
int    feupdateenv(const fenv_t *);
```

The **FENV_ACCESS** pragma provides a means to inform the implementation when an application might access the floating-point environment to test floating-point status flags or run under non-default floating-point control modes. The pragma shall occur either outside external declarations or preceding all explicit declarations and statements inside a compound statement. When outside external declarations, the pragma takes effect from its occurrence until another **FENV_ACCESS** pragma is encountered, or until the end of the translation unit. When inside a compound statement, the pragma takes effect from its occurrence until another **FENV_ACCESS** pragma is encountered (including within a nested compound statement), or until the end of the compound statement; at the end of a compound statement the state for the pragma is restored to its condition just before the compound statement. If this pragma is used in any other context, the behavior is undefined. If part of an application tests floating-point status flags, sets floating-point control modes, or runs under non-default mode settings, but was translated with the state for the **FENV_ACCESS** pragma off, the behavior is undefined. The default state (on or off) for the pragma is implementation-defined. (When execution passes from a part of the application translated with **FENV_ACCESS** off to a part translated with **FENV_ACCESS** on, the state of the floating-point status flags is unspecified and the floating-point control modes have their default settings.)

APPLICATION USAGE

This header is designed to support the floating-point exception status flags and directed-rounding control modes required by the IEC 60559:1989 standard, and other similar floating-point state information. Also it is designed to facilitate code portability among all systems.

Certain application programming conventions support the intended model of use for the floating-point environment:

- A function call does not alter its caller's floating-point control modes, clear its caller's floating-point status flags, nor depend on the state of its caller's floating-point status flags unless the function is so documented.
- A function call is assumed to require default floating-point control modes, unless its documentation promises otherwise.

Headers

<fenv.h>

- A function call is assumed to have the potential for raising floating-point exceptions, unless its documentation promises otherwise.

With these conventions, an application can safely assume default floating-point control modes (or be unaware of them). The responsibilities associated with accessing the floating-point environment fall on the application that does so explicitly.

Even though the rounding direction macros may expand to constants corresponding to the values of FLT_ROUNDS, they are not required to do so.

For example:

```
#include <fenv.h>
void f(double x)
{
    #pragma STDC FENV_ACCESS ON
    void g(double);
    void h(double);
    /* ... */
    g(x + 1);
    h(x + 1);
    /* ... */
}
```

If the function *g()* might depend on status flags set as a side-effect of the first *x+1*, or if the second *x+1* might depend on control modes set as a side-effect of the call to function *g()*, then the application shall contain an appropriately placed invocation as follows:

```
#pragma STDC FENV_ACCESS ON
```

RATIONALE

The **fexcept_t** Type

fexcept_t does not have to be an integer type. Its values must be obtained by a call to *fegetexceptflag()*, and cannot be created by logical operations from the exception macros. An implementation might simply implement **fexcept_t** as an **int** and use the representations reflected by the exception macros, but is not required to; other representations might contain extra information about the exceptions. **fexcept_t** might be a **struct** with a member for each exception (that might hold the address of the first or last floating-point instruction that caused that exception). The ISO/IEC 9899:1999 standard makes no claims about the internals of an **fexcept_t**, and so the user cannot inspect it.

Exception and Rounding Macros

Macros corresponding to unsupported modes and rounding directions are not defined by the implementation and must not be defined by the application. An application might use **#ifdef** to test for this.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *feclearexcept()*, *fegetenv()*, *fegetexceptflag()*, *fegetround()*, *fehldexcept()*, *feraiseexcept()*, *fetestexcept()*, *feupdateenv()*

CHANGE HISTORY

First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

The return types for *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, *fegetenv()*, *fesetenv()*, and *feupdateenv()* are changed from **void** to **int** for alignment with the ISO/IEC 9899: 1999 standard, Defect Report 202.

Issue 7

ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #37 (SD5-XBD-ERN-49) is applied.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 3 #36 is applied.

SD5-XBD-ERN-48 and SD5-XBD-ERN-69 are applied.

This reference page is clarified with respect to macros and symbolic constants.

Headers

<float.h>

NAME

float.h — floating types

SYNOPSIS

#include <float.h>

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The characteristics of floating types are defined in terms of a model that describes a representation of floating-point numbers and values that provide information about an implementation's floating-point arithmetic.

The following parameters are used to define the model for each floating-point type:

s Sign (± 1).

b Base or radix of exponent representation (an integer > 1).

e Exponent (an integer between a minimum $e_{\min}$ and a maximum $e_{\max}$).

p Precision (the number of base-*b* digits in the significand).

f_k Non-negative integers less than *b* (the significand digits).

A floating-point number *x* is defined by the following model:

$$x = sb^e \sum_{k=1}^p f_k b^{-k}, e_{\min} \leq e \leq e_{\max}$$

In addition to normalized floating-point numbers ($f_1 > 0$ if $x \neq 0$), floating types may be able to contain other kinds of floating-point numbers, such as subnormal floating-point numbers ($x \neq 0$, $e = e_{\min}$, $f_1 = 0$) and unnormalized floating-point numbers ($x \neq 0$, $e > e_{\min}$, $f_1 = 0$), and values that are not floating-point numbers, such as infinities and NaNs. A NaN is an encoding signifying Not-a-Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a floating-point exception; a *signaling NaN* generally raises a floating-point exception when occurring as an arithmetic operand.

An implementation may give zero and non-numeric values, such as infinities and NaNs, a sign, or may leave them unsigned. Wherever such values are unsigned, any requirement in POSIX.1-2008 to retrieve the sign shall produce an unspecified sign and any requirement to set the sign shall be ignored.

The accuracy of the floating-point operations ('+', '-', '*', '/') and of the functions in <math.h> and <complex.h> that return floating-point results is implementation-defined, as is the accuracy of the conversion between floating-point internal representations and string representations performed by the functions in <stdio.h>, <stdlib.h>, and <wchar.h>. The implementation may state that the accuracy is unknown.

All integer values in the <float.h> header, except FLT_ROUNDS, shall be constant expressions suitable for use in #if preprocessing directives; all floating values shall be constant expressions. All except DECIMAL_DIG, FLT_EVAL_METHOD, FLT_RADIX, and FLT_ROUNDS have separate names for all three floating-point types. The floating-point model representation is provided for all values except FLT_EVAL_METHOD and FLT_ROUNDS.

The rounding mode for floating-point addition is characterized by the implementation-defined

<float.h>

Headers

8146 value of FLT_ROUNDS:

- 8147 -1 Indeterminable.
 8148 0 Toward zero.
 8149 1 To nearest.
 8150 2 Toward positive infinity.
 8151 3 Toward negative infinity.

8152 All other values for FLT_ROUNDS characterize implementation-defined rounding behavior.

8153 The values of operations with floating operands and values subject to the usual arithmetic
 8154 conversions and of floating constants are evaluated to a format whose range and precision may
 8155 be greater than required by the type. The use of evaluation formats is characterized by the
 8156 implementation-defined value of FLT_EVAL_METHOD:

- 8157 -1 Indeterminable.
 8158 0 Evaluate all operations and constants just to the range and precision of the type.
 8159 1 Evaluate operations and constants of type **float** and **double** to the range and precision of
 8160 the **double** type; evaluate **long double** operations and constants to the range and precision
 8161 of the **long double** type.
 8162 2 Evaluate all operations and constants to the range and precision of the **long double** type.

8163 All other negative values for FLT_EVAL_METHOD characterize implementation-defined
 8164 behavior.

8165 The <float.h> header shall define the following values as constant expressions with
 8166 implementation-defined values that are greater or equal in magnitude (absolute value) to those
 8167 shown, with the same sign.

- 8168 • Radix of exponent representation, b .
 8169 FLT_RADIX 2
- 8170 • Number of base-FLT_RADIX digits in the floating-point significand, p .
 8171 FLT_MANT_DIG
 8172 DBL_MANT_DIG
 8173 LDBL_MANT_DIG
- 8174 • Number of decimal digits, n , such that any floating-point number in the widest supported
 8175 floating type with $p_{\max}$ radix b digits can be rounded to a floating-point number with n
 8176 decimal digits and back again without change to the value.

$$\left\{ \begin{array}{ll} p_{\max} \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \left\lceil 1 + p_{\max} \log_{10} b \right\rceil & \text{otherwise} \end{array} \right.$$

8177 DECIMAL_DIG 10

Headers

<float.h>

- 8178 • Number of decimal digits, q , such that any floating-point number with q decimal digits can
 8179 be rounded into a floating-point number with p radix b digits and back again without
 8180 change to the q decimal digits.

$$\begin{cases} p \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \lfloor (p-1) \log_{10} b \rfloor & \text{otherwise} \end{cases}$$

8181 FLT_DIG 6

8182 DBL_DIG 10

8183 LDBL_DIG 10

- 8184 • Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a
 8185 normalized floating-point number, $e_{\min}$.

8186 FLT_MIN_EXP

8187 DBL_MIN_EXP

8188 LDBL_MIN_EXP

- 8189 • Minimum negative integer such that 10 raised to that power is in the range of normalized
 8190 floating-point numbers.

$$\left\lceil \log_{10} b^{e_{\min}-1} \right\rceil$$

8191 FLT_MIN_10_EXP -37

8192 DBL_MIN_10_EXP -37

8193 LDBL_MIN_10_EXP -37

- 8194 • Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable
 8195 finite floating-point number, $e_{\max}$.

8196 FLT_MAX_EXP

8197 DBL_MAX_EXP

8198 LDBL_MAX_EXP

- 8199 • Maximum integer such that 10 raised to that power is in the range of representable finite
 8200 floating-point numbers.

$$\left\lceil \log_{10} ((1 - b^{-p}) b^{e_{\max}}) \right\rceil$$

8201 FLT_MAX_10_EXP +37

8202 DBL_MAX_10_EXP +37

<float.h>

Headers

8203 LDBL_MAX_10_EXP +37

8204 The <float.h> header shall define the following values as constant expressions with
8205 implementation-defined values that are greater than or equal to those shown:

- 8206
- Maximum representable finite floating-point number.

$$(1 - b^{-p}) b^{e_{\max}}$$

8207 FLT_MAX 1E+37

8208 DBL_MAX 1E+37

8209 LDBL_MAX 1E+37

8210 The <float.h> header shall define the following values as constant expressions with
8211 implementation-defined (positive) values that are less than or equal to those shown:

- 8212
- The difference between 1 and the least value greater than 1 that is representable in the
8213 given floating-point type, b^{1-p} .

8214 FLT_EPSILON 1E-5

8215 DBL_EPSILON 1E-9

8216 LDBL_EPSILON 1E-9

- 8217
- Minimum normalized positive floating-point number, $b^{e_{\min}-1}$.

8218 FLT_MIN 1E-37

8219 DBL_MIN 1E-37

8220 LDBL_MIN 1E-37

8221 APPLICATION USAGE

8222 None.

8223 RATIONALE

8224 None.

8225 FUTURE DIRECTIONS

8226 None.

8227 SEE ALSO

8228 <complex.h>, <math.h>, <stdio.h>, <stdlib.h>, <wchar.h>

8229 CHANGE HISTORY

8230 First released in Issue 4. Derived from the ISO C standard.

8231 Issue 6

8232 The description of the operations with floating-point values is updated for alignment with the
8233 ISO/IEC 9899:1999 standard.

8234 Issue 7

8235 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #4 (SD5-XBD-ERN-50) and #5
8236 (SD5-XBD-ERN-51) are applied.

Headers

<fmtmsg.h>

NAME

fmtmsg.h — message display structures

SYNOPSIS

```
#include <fmtmsg.h>
```

DESCRIPTION

The <fmtmsg.h> header shall define the following symbolic constants:

MM_HARD	Source of the condition is hardware.
MM_SOFT	Source of the condition is software.
MM_FIRM	Source of the condition is firmware.
MM_APPL	Condition detected by application.
MM_UTIL	Condition detected by utility.
MM_OPSYS	Condition detected by operating system.
MM_RECOVER	Recoverable error.
MM_NRECOV	Non-recoverable error.
MM_HALT	Error causing application to halt.
MM_ERROR	Application has encountered a non-fatal fault.
MM_WARNING	Application has detected unusual non-error condition.
MM_INFO	Informative message.
MM_NOSEV	No severity level provided for the message.
MM_PRINT	Display message on standard error.
MM_CONSOLE	Display message on system console.

The table below indicates the null values and identifiers for *fmtmsg()* arguments. The <fmtmsg.h> header shall define the symbolic constants in the **Identifier** column, which shall have the type indicated in the **Type** column:

Argument	Type	Null-Value	Identifier
<i>label</i>	char *	(char*)0	MM_NULLLBL
<i>severity</i>	int	0	MM_NULLSEV
<i>class</i>	long	0L	MM_NULLMC
<i>text</i>	char *	(char*)0	MM_NULLTXT
<i>action</i>	char *	(char*)0	MM_NULLACT
<i>tag</i>	char *	(char*)0	MM_NULLTAG

The <fmtmsg.h> header shall also define the following symbolic constants for use as return values for *fmtmsg()*:

MM_OK	The function succeeded.
MM_NOTOK	The function failed completely.
MM_NOMSG	The function was unable to generate a message on standard error, but otherwise succeeded.

<fmtmsg.h>

Headers

8274 MM_NOCON The function was unable to generate a console message, but otherwise
8275 succeeded.

8276 The following shall be declared as a function and may also be defined as a macro. A function
8277 prototype shall be provided.

8278 int fmtmsg(long, const char *, int,
8279 const char *, const char *, const char *);

8280 APPLICATION USAGE

8281 None.

8282 RATIONALE

8283 None.

8284 FUTURE DIRECTIONS

8285 None.

8286 SEE ALSO

8287 XSH *fmtmsg()*

8288 CHANGE HISTORY

8289 First released in Issue 4, Version 2.

8290 Issue 7

8291 This reference page is clarified with respect to macros and symbolic constants.

Headers

<fnmatch.h>

NAME

fnmatch.h — filename-matching types

SYNOPSIS

```
#include <fnmatch.h>
```

DESCRIPTION

The <fnmatch.h> header shall define the following symbolic constants:

FNM_NOMATCH The string does not match the specified pattern.

FNM_PATHNAME <slash> in *string* only matches <slash> in *pattern*.

FNM_PERIOD Leading <period> in *string* must be exactly matched by <period> in *pattern*.

FNM_NOESCAPE Disable backslash escaping.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
int fnmatch(const char *, const char *, int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *fnmatch()*

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 6

The FNM_NOSYS constant is marked obsolescent.

Issue 7

The obsolescent FNM_NOSYS constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

<ftw.h>

Headers

8321 NAME

8322 ftw.h — file tree traversal

8323 SYNOPSIS

8324 XSI `#include <ftw.h>`

8325 DESCRIPTION

8326 The <ftw.h> header shall define the **FTW** structure, which shall include at least the following
8327 members:8328 `int base`
8329 `int level`8330 The <ftw.h> header shall define the following symbolic constants for use as values of the third
8331 argument to the application-supplied function that is passed as the second argument to *ftw()*
8332 and *nftw()*:8333 **FTW_F** File.
8334 **FTW_D** Directory.
8335 **FTW_DNR** Directory without read permission.
8336 **FTW_DP** Directory with subdirectories visited.
8337 **FTW_NS** Unknown type; *stat()* failed.
8338 **FTW_SL** Symbolic link.
8339 **FTW_SLN** Symbolic link that names a nonexistent file.8340 The <ftw.h> header shall define the following symbolic constants for use as values of the fourth
8341 argument to *nftw()*:8342 **FTW_PHYS** Physical walk, does not follow symbolic links. Otherwise, *nftw()* follows
8343 links but does not walk down any path that crosses itself.
8344 **FTW_MOUNT** The walk does not cross a mount point.
8345 **FTW_DEPTH** All subdirectories are visited before the directory itself.
8346 **FTW_CHDIR** The walk changes to each directory before reading it.8347 The following shall be declared as functions and may also be defined as macros. Function
8348 prototypes shall be provided.8349 OB `int ftw(const char *, int (*)(const char *, const struct stat *,`
8350 `int), int);`
8351 `int nftw(const char *, int (*)(const char *, const struct stat *,`
8352 `int, struct FTW *), int, int);`8353 The <ftw.h> header shall define the **stat** structure and the symbolic names for *st_mode* and the
8354 file type test macros as described in <sys/stat.h>.

8355 Inclusion of the <ftw.h> header may also make visible all symbols from <sys/stat.h>.

Headers

<ftw.h>

8356 **APPLICATION USAGE**

8357 None.

8358 **RATIONALE**

8359 None.

8360 **FUTURE DIRECTIONS**

8361 None.

8362 **SEE ALSO**

8363 <sys/stat.h>

8364 XSH *ftw()*, *nftw()*8365 **CHANGE HISTORY**

8366 First released in Issue 1. Derived from Issue 1 of the SVID.

8367 **Issue 5**

8368 A description of FTW_DP is added.

8369 **Issue 7**8370 The *ftw()* function is marked obsolescent.

8371 This reference page is clarified with respect to macros and symbolic constants.

<glob.h>*Headers*8372 **NAME**

8373 glob.h — pathname pattern-matching types

8374 **SYNOPSIS**

8375 #include <glob.h>

8376 **DESCRIPTION**8377 The **<glob.h>** header shall define the structures and symbolic constants used by the *glob()*
8378 function.8379 The **<glob.h>** header shall define the **glob_t** structure type, which shall include at least the
8380 following members:

8381 size_t gl_pathc Count of paths matched by *pattern*.
 8382 char **gl_pathv Pointer to a list of matched pathnames.
 8383 size_t gl_offs Slots to reserve at the beginning of *gl_pathv*.

8384 The **<glob.h>** header shall define the **size_t** type as described in **<sys/types.h>**.8385 The **<glob.h>** header shall define the following symbolic constants as values for the *flags*
8386 argument:

8387 GLOB_APPEND Append generated pathnames to those previously obtained.
 8388 GLOB_DOOFFS Specify how many null pointers to add to the beginning of *gl_pathv*.
 8389 GLOB_ERR Cause *glob()* to return on error.
 8390 GLOB_MARK Each pathname that is a directory that matches *pattern* has a <slash>
 8391 appended.
 8392 GLOB_NOCHECK If *pattern* does not match any pathname, then return a list consisting of
 8393 only *pattern*.
 8394 GLOB_NOESCAPE Disable backslash escaping.
 8395 GLOB_NOSORT Do not sort the pathnames returned.
 8396 The **<glob.h>** header shall define the following symbolic constants as error return values:
 8397 GLOB_ABORTED The scan was stopped because GLOB_ERR was set or (*errfunc)()
 8398 returned non-zero.
 8399 GLOB_NOMATCH The pattern does not match any existing pathname, and
 8400 GLOB_NOCHECK was not set in *flags*.
 8401 GLOB_NOSPACE An attempt to allocate memory failed.

8402 The following shall be declared as functions and may also be defined as macros. Function
8403 prototypes shall be provided.

8404 int glob(const char *restrict, int, int (*)(const char *, int),
 8405 glob_t *restrict);
 8406 void globfree(glob_t *);

Headers

<glob.h>

8407 **APPLICATION USAGE**

8408 None.

8409 **RATIONALE**

8410 None.

8411 **FUTURE DIRECTIONS**

8412 None.

8413 **SEE ALSO**

8414 <sys/types.h>

8415 XSH *glob()*8416 **CHANGE HISTORY**

8417 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8418 **Issue 6**8419 The **restrict** keyword is added to the prototype for *glob()*.

8420 The GLOB_NOSYS constant is marked obsolescent.

8421 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/8 is applied, correcting the *glob()*
8422 prototype definition by removing the **restrict** qualifier from the function pointer argument.8423 **Issue 7**8424 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size_t**

8425 The obsolescent GLOB_NOSYS constant is removed.

8426 This reference page is clarified with respect to macros and symbolic constants.

<grp.h>

Headers

8427 **NAME**

8428 grp.h — group structure

8429 **SYNOPSIS**

8430 #include <grp.h>

8431 **DESCRIPTION**8432 The **<grp.h>** header shall declare the **group** structure, which shall include the following
8433 members:

8434 char *gr_name The name of the group.
 8435 gid_t gr_gid Numerical group ID.
 8436 char **gr_mem Pointer to a null-terminated array of character
 8437 pointers to member names.

8438 The **<grp.h>** header shall define the **gid_t** and **size_t** types as described in **<sys/types.h>**.8439 The following shall be declared as functions and may also be defined as macros. Function
8440 prototypes shall be provided.

```

8441       void            endgrent(void);
8442       struct group    *getgrent(void);
8443       struct group    *getgrgid(gid_t);
8444       int             getgrgid_r(gid_t, struct group *, char *,
8445                                 size_t, struct group **);
8446       struct group    *getgrnam(const char *);
8447       int             getgrnam_r(const char *, struct group *, char *,
8448                                 size_t, struct group **);
8449       XSI     void     setgrent(void);

```

8450 **APPLICATION USAGE**

8451 None.

8452 **RATIONALE**

8453 None.

8454 **FUTURE DIRECTIONS**

8455 None.

8456 **SEE ALSO**8457 [<sys/types.h>](#)8458 XSH [endgrent\(\)](#), [getgrgid\(\)](#), [getgrnam\(\)](#)8459 **CHANGE HISTORY**

8460 First released in Issue 1.

8461 **Issue 5**

8462 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8463 **Issue 6**8464 The following new requirements on POSIX implementations derive from alignment with the
8465 Single UNIX Specification:

- 8466 • The definition of **gid_t** is mandated.
- 8467 • The [getgrgid_r\(\)](#) and [getgrnam_r\(\)](#) functions are marked as part of the Thread-Safe
8468 Functions option.

*Headers***<grp.h>**8469
8470**Issue 7**SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

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

<iconv.h>

Headers

8471 **NAME**

8472 iconv.h — codeset conversion facility

8473 **SYNOPSIS**

8474 #include <iconv.h>

8475 **DESCRIPTION**

8476 The <iconv.h> header shall define the following types:

8477 **iconv_t** Identifies the conversion from one codeset to another.8478 **size_t** As described in <sys/types.h>.8479 The following shall be declared as functions and may also be defined as macros. Function
8480 prototypes shall be provided.

```

8481 size_t iconv(iconv_t, char **restrict, size_t *restrict,
8482             char **restrict, size_t *restrict);
8483 int iconv_close(iconv_t);
8484 iconv_t iconv_open(const char *, const char *);

```

8485 **APPLICATION USAGE**

8486 None.

8487 **RATIONALE**

8488 None.

8489 **FUTURE DIRECTIONS**

8490 None.

8491 **SEE ALSO**

8492 <sys/types.h>

8493 XSH *iconv()*, *iconv_close()*, *iconv_open()*8494 **CHANGE HISTORY**

8495 First released in Issue 4.

8496 **Issue 6**8497 The **restrict** keyword is added to the prototype for *iconv()*.8498 **Issue 7**8499 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size_t** type.

8500 The <iconv.h> header is moved from the XSI option to the Base.

Headers

<inttypes.h>

NAME

inttypes.h — fixed size integer types

SYNOPSIS

```
#include <inttypes.h>
```

DESCRIPTION

Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to enable the visibility of these symbols in this header.

The <inttypes.h> header shall include the <stdint.h> header.

The <inttypes.h> header shall define at least the following type:

imaxdiv_t Structure type that is the type of the value returned by the *imaxdiv()* function.

The <inttypes.h> header shall define the following macros. Each expands to a character string literal containing a conversion specifier, possibly modified by a length modifier, suitable for use within the *format* argument of a formatted input/output function when converting the corresponding integer type. These macros have the general form of PRI (character string literals for the *fprintf()* and *fwprintf()* family of functions) or SCN (character string literals for the *fscanf()* and *fwscanf()* family of functions), followed by the conversion specifier, followed by a name corresponding to a similar type name in <stdint.h>. In these names, *N* represents the width of the type as described in <stdint.h>. For example, *PRIdFAST32* can be used in a format string to print the value of an integer of type *int_fast32_t*.

The *fprintf()* macros for signed integers are:

<i>PRIdN</i>	<i>PRIdLEASTN</i>	<i>PRIdFASTN</i>	<i>PRIdMAX</i>	<i>PRIdPTR</i>
<i>PRIdN</i>	<i>PRIdLEASTN</i>	<i>PRIdFASTN</i>	<i>PRIdMAX</i>	<i>PRIdPTR</i>

The *fprintf()* macros for unsigned integers are:

<i>PRIoN</i>	<i>PRIoLEASTN</i>	<i>PRIoFASTN</i>	<i>PRIoMAX</i>	<i>PRIoPTR</i>
<i>PRIoN</i>	<i>PRIoLEASTN</i>	<i>PRIoFASTN</i>	<i>PRIoMAX</i>	<i>PRIoPTR</i>
<i>PRIxN</i>	<i>PRIxLEASTN</i>	<i>PRIxFASTN</i>	<i>PRIxMAX</i>	<i>PRIxPTR</i>
<i>PRIXN</i>	<i>PRIXLEASTN</i>	<i>PRIXFASTN</i>	<i>PRIXMAX</i>	<i>PRIXPTR</i>

The *fscanf()* macros for signed integers are:

<i>SCNdN</i>	<i>SCNdLEASTN</i>	<i>SCNdFASTN</i>	<i>SCNdMAX</i>	<i>SCNdPTR</i>
<i>SCNiN</i>	<i>SCNiLEASTN</i>	<i>SCNiFASTN</i>	<i>SCNiMAX</i>	<i>SCNiPTR</i>

The *fscanf()* macros for unsigned integers are:

<i>SCNoN</i>	<i>SCNoLEASTN</i>	<i>SCNoFASTN</i>	<i>SCNoMAX</i>	<i>SCNoPTR</i>
<i>SCNuN</i>	<i>SCNuLEASTN</i>	<i>SCNuFASTN</i>	<i>SCNuMAX</i>	<i>SCNuPTR</i>
<i>SCNxN</i>	<i>SCNxLEASTN</i>	<i>SCNxFASTN</i>	<i>SCNxMAX</i>	<i>SCNxPTR</i>

For each type that the implementation provides in <stdint.h>, the corresponding *fprintf()* and *fwprintf()* macros shall be defined and the corresponding *fscanf()* and *fwscanf()* macros shall be defined unless the implementation does not have a suitable modifier for the type.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
intmax_t imaxabs(intmax_t);
imaxdiv_t imaxdiv(intmax_t, intmax_t);
```

<inttypes.h>*Headers*

```

8543     intmax_t  strtoumax(const char *restrict, char **restrict, int);
8544     uintmax_t strtoumax(const char *restrict, char **restrict, int);
8545     intmax_t  wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);
8546     uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);

```

EXAMPLES

```

8547     #include <inttypes.h>
8548     #include <wchar.h>
8549     int main(void)
8550     {
8551         uintmax_t i = UINTMAX_MAX; // This type always exists.
8552         wprintf(L"The largest integer value is %020"
8553             PRIxMAX "\n", i);
8554         return 0;
8555     }
8556

```

APPLICATION USAGE

8557 The purpose of **<inttypes.h>** is to provide a set of integer types whose definitions are consistent
8558 across machines and independent of operating systems and other implementation
8559 idiosyncrasies. It defines, through **typedef**, integer types of various sizes. Implementations are
8560 free to **typedef** them as ISO C standard integer types or extensions that they support. Consistent
8561 use of this header will greatly increase the portability of applications across platforms.

RATIONALE

8563 The ISO/IEC 9899:1990 standard specified that the language should support four signed and
8564 unsigned integer data types—**char**, **short**, **int**, and **long**—but placed very little requirement on
8565 their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int** and
8566 not smaller than 32 bits. For 16-bit systems, most implementations assigned 8, 16, 16, and 32 bits
8567 to **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice has been to
8568 assign 8, 16, 32, and 32 bits to these types. This difference in **int** size can create some problems
8569 for users who migrate from one system to another which assigns different sizes to integer types,
8570 because the ISO C standard integer promotion rule can produce silent changes unexpectedly.
8571 The need for defining an extended integer type increased with the introduction of 64-bit
8572 systems.

FUTURE DIRECTIONS

8574 Macro names beginning with PRI or SCN followed by any lowercase letter or 'X' may be added
8575 to the macros defined in the **<inttypes.h>** header.

SEE ALSO

8577 XSH Section 2.2 (on page 468), *imaxabs()*, *imaxdiv()*, *strtoumax()*, *wcstoumax()*

CHANGE HISTORY

8579 First released in Issue 5.

Issue 6

8581 The Open Group Base Resolution bwg97-006 is applied.

8582 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

Headers

<iso646.h>

NAME

iso646.h — alternative spellings

SYNOPSIS

```
#include <iso646.h>
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The <iso646.h> header shall define the following eleven macros (on the left) that expand to the corresponding tokens (on the right):

and &&

and_eq &=

bitand &

bitor |

compl ~

not !

not_eq !=

or ||

or_eq |=

xor ^

xor_eq ^=

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

<langinfo.h>

Headers

8615 NAME

8616 langinfo.h — language information constants

8617 SYNOPSIS

8618 #include <langinfo.h>

8619 DESCRIPTION

8620 The <langinfo.h> header shall define the symbolic constants used to identify items of *langinfo* data (see *nl_langinfo()*).8621 The <langinfo.h> header shall define the **locale_t** type as described in <locale.h>.8622 The <langinfo.h> header shall define the **nl_item** type as described in <nl/types.h>.8623 The <langinfo.h> header shall define the following symbolic constants with type **nl_item**. The entries under **Category** indicate in which *setlocale()* category each item is defined.

Constant	Category	Meaning
CODESET	LC_CTYPE	Codeset name.
D_T_FMT	LC_TIME	String for formatting date and time.
D_FMT	LC_TIME	Date format string.
T_FMT	LC_TIME	Time format string.
T_FMT_AMPM	LC_TIME	a.m. or p.m. time format string.
AM_STR	LC_TIME	Ante-meridiem affix.
PM_STR	LC_TIME	Post-meridiem affix.
DAY_1	LC_TIME	Name of the first day of the week (for example, Sunday).
DAY_2	LC_TIME	Name of the second day of the week (for example, Monday).
DAY_3	LC_TIME	Name of the third day of the week (for example, Tuesday).
DAY_4	LC_TIME	Name of the fourth day of the week (for example, Wednesday).
DAY_5	LC_TIME	Name of the fifth day of the week (for example, Thursday).
DAY_6	LC_TIME	Name of the sixth day of the week (for example, Friday).
DAY_7	LC_TIME	Name of the seventh day of the week (for example, Saturday).
ABDAY_1	LC_TIME	Abbreviated name of the first day of the week.
ABDAY_2	LC_TIME	Abbreviated name of the second day of the week.
ABDAY_3	LC_TIME	Abbreviated name of the third day of the week.
ABDAY_4	LC_TIME	Abbreviated name of the fourth day of the week.
ABDAY_5	LC_TIME	Abbreviated name of the fifth day of the week.
ABDAY_6	LC_TIME	Abbreviated name of the sixth day of the week.
ABDAY_7	LC_TIME	Abbreviated name of the seventh day of the week.
MON_1	LC_TIME	Name of the first month of the year.
MON_2	LC_TIME	Name of the second month.
MON_3	LC_TIME	Name of the third month.
MON_4	LC_TIME	Name of the fourth month.
MON_5	LC_TIME	Name of the fifth month.
MON_6	LC_TIME	Name of the sixth month.
MON_7	LC_TIME	Name of the seventh month.
MON_8	LC_TIME	Name of the eighth month.
MON_9	LC_TIME	Name of the ninth month.
MON_10	LC_TIME	Name of the tenth month.
MON_11	LC_TIME	Name of the eleventh month.
MON_12	LC_TIME	Name of the twelfth month.
ABMON_1	LC_TIME	Abbreviated name of the first month.

Headers

<langinfo.h>

Constant	Category	Meaning
ABMON_2	LC_TIME	Abbreviated name of the second month.
ABMON_3	LC_TIME	Abbreviated name of the third month.
ABMON_4	LC_TIME	Abbreviated name of the fourth month.
ABMON_5	LC_TIME	Abbreviated name of the fifth month.
ABMON_6	LC_TIME	Abbreviated name of the sixth month.
ABMON_7	LC_TIME	Abbreviated name of the seventh month.
ABMON_8	LC_TIME	Abbreviated name of the eighth month.
ABMON_9	LC_TIME	Abbreviated name of the ninth month.
ABMON_10	LC_TIME	Abbreviated name of the tenth month.
ABMON_11	LC_TIME	Abbreviated name of the eleventh month.
ABMON_12	LC_TIME	Abbreviated name of the twelfth month.
ERA	LC_TIME	Era description segments.
ERA_D_FMT	LC_TIME	Era date format string.
ERA_D_T_FMT	LC_TIME	Era date and time format string.
ERA_T_FMT	LC_TIME	Era time format string.
ALT_DIGITS	LC_TIME	Alternative symbols for digits.
RADIXCHAR	LC_NUMERIC	Radix character.
THOUSEP	LC_NUMERIC	Separator for thousands.
YESEXPR	LC_MESSAGES	Affirmative response expression.
NOEXPR	LC_MESSAGES	Negative response expression.
CRNCYSTR	LC_MONETARY	Local currency symbol, preceded by '-' if the symbol should appear before the value, '+' if the symbol should appear after the value, or '.' if the symbol should replace the radix character. If the local currency symbol is the empty string, implementations may return the empty string ("").

If the locale's values for **p_cs_precedes** and **n_cs_precedes** do not match, the value of *nl_langinfo*(CRNCYSTR) and *nl_langinfo_l*(CRNCYSTR, loc) is unspecified.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided:

```
char *nl_langinfo(nl_item);
char *nl_langinfo_l(nl_item, locale_t);
```

Inclusion of the <langinfo.h> header may also make visible all symbols from <nl_types.h>.

APPLICATION USAGE

Wherever possible, users are advised to use functions compatible with those in the ISO C standard to access items of *langinfo* data. In particular, the *strftime*() function should be used to access date and time information defined in category *LC_TIME*. The *localeconv*() function should be used to access information corresponding to *RADIXCHAR*, *THOUSEP*, and *CRNCYSTR*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 7 (on page 135), <locale.h>, <nl_types.h>

XSH *nl_langinfo*(), *localeconv*(), *strfmon*(), *strftime*()

<langinfo.h>

Headers

8709 **CHANGE HISTORY**

8710 First released in Issue 2.

8711 **Issue 5**

8712 The constants YESSTR and NOSTR are marked LEGACY.

8713 **Issue 6**

8714 The constants YESSTR and NOSTR are removed.

8715 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/9 is applied, adding a sentence to
 8716 the “Meaning” column entry for the CRNCYSTR constant. This change is to accommodate
 8717 historic practice.

8718 **Issue 7**

8719 The <langinfo.h> header is moved from the XSI option to the Base.

8720 The `nl_langinfo_l()` function is added from The Open Group Technical Standard, 2006, Extended
 8721 API Set Part 4.

8722 This reference page is clarified with respect to macros and symbolic constants, and a declaration
 8723 for the `locale_t` type is added.

Headers

<libgen.h>

NAME

libgen.h — definitions for pattern matching functions

SYNOPSIS

```
#include <libgen.h>
```

DESCRIPTION

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
char *basename(char *);
```

```
char *dirname(char *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *basename()*, *dirname()*

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

The function prototypes for *basename()* and *dirname()* are changed to indicate that the first argument is of type **char *** rather than **const char ***.

Issue 6

The **__loc1** symbol and the *regcmp()* and *regex()* functions are removed.

<limits.h>*Headers***NAME**

limits.h — implementation-defined constants

SYNOPSIS

```
#include <limits.h>
```

DESCRIPTION

Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to enable the visibility of these symbols in this header.

Many of the symbols listed here are not defined by the ISO/IEC 9899:1999 standard. Such symbols are not shown as CX shaded.

The **<limits.h>** header shall define macros and symbolic constants for various limits. Different categories of limits are described below, representing various limits on resources that the implementation imposes on applications. All macros and symbolic constants defined in this header shall be suitable for use in **#if** preprocessing directives.

Implementations may choose any appropriate value for each limit, provided it is not more restrictive than the Minimum Acceptable Values listed below. Symbolic constant names beginning with **_POSIX** may be found in **<unistd.h>**.

Applications should not assume any particular value for a limit. To achieve maximum portability, an application should not require more resource than the Minimum Acceptable Value quantity. However, an application wishing to avail itself of the full amount of a resource available on an implementation may make use of the value given in **<limits.h>** on that particular implementation, by using the macros and symbolic constants listed below. It should be noted, however, that many of the listed limits are not invariant, and at runtime, the value of the limit may differ from those given in this header, for the following reasons:

- The limit is pathname-dependent.
- The limit differs between the compile and runtime machines.

For these reasons, an application may use the *fpathconf()*, *pathconf()*, and *sysconf()* functions to determine the actual value of a limit at runtime.

The items in the list ending in **_MIN** give the most negative values that the mathematical types are guaranteed to be capable of representing. Numbers of a more negative value may be supported on some implementations, as indicated by the **<limits.h>** header on the implementation, but applications requiring such numbers are not guaranteed to be portable to all implementations. For positive constants ending in **_MIN**, this indicates the minimum acceptable value.

Runtime Invariant Values (Possibly Indeterminate)

A definition of one of the symbolic constants in the following list shall be omitted from **<limits.h>** on specific implementations where the corresponding value is equal to or greater than the stated minimum, but is unspecified.

This indetermination might depend on the amount of available memory space on a specific instance of a specific implementation. The actual value supported by a specific instance shall be provided by the *sysconf()* function.

{AIO_LISTIO_MAX}

Maximum number of I/O operations in a single list I/O call supported by the implementation.

Minimum Acceptable Value: {_POSIX_AIO_LISTIO_MAX}

Headers

<limits.h>

8793		{AIO_MAX}
8794		Maximum number of outstanding asynchronous I/O operations supported by the implementation.
8795		Minimum Acceptable Value: {_POSIX_AIO_MAX}
8796		
8797		{AIO_PRIO_DELTA_MAX}
8798		The maximum amount by which a process can decrease its asynchronous I/O priority level from its own scheduling priority.
8799		Minimum Acceptable Value: 0
8800		
8801		{ARG_MAX}
8802		Maximum length of argument to the <i>exec</i> functions including environment data.
8803		Minimum Acceptable Value: {_POSIX_ARG_MAX}
8804		{ATEXIT_MAX}
8805		Maximum number of functions that may be registered with <i>atexit()</i> .
8806		Minimum Acceptable Value: 32
8807		{CHILD_MAX}
8808		Maximum number of simultaneous processes per real user ID.
8809		Minimum Acceptable Value: {_POSIX_CHILD_MAX}
8810		{DELAYTIMER_MAX}
8811		Maximum number of timer expiration overruns.
8812		Minimum Acceptable Value: {_POSIX_DELAYTIMER_MAX}
8813		{HOST_NAME_MAX}
8814		Maximum length of a host name (not including the terminating null) as returned from the <i>gethostname()</i> function.
8815		Minimum Acceptable Value: {_POSIX_HOST_NAME_MAX}
8816		
8817	XSI	{IOV_MAX}
8818		Maximum number of iovec structures that one process has available for use with <i>readv()</i> or <i>writev()</i> .
8819		Minimum Acceptable Value: {_XOPEN_IOV_MAX}
8820		
8821		{LOGIN_NAME_MAX}
8822		Maximum length of a login name.
8823		Minimum Acceptable Value: {_POSIX_LOGIN_NAME_MAX}
8824	MSG	{MQ_OPEN_MAX}
8825		The maximum number of open message queue descriptors a process may hold.
8826		Minimum Acceptable Value: {_POSIX_MQ_OPEN_MAX}
8827	MSG	{MQ_PRIO_MAX}
8828		The maximum number of message priorities supported by the implementation.
8829		Minimum Acceptable Value: {_POSIX_MQ_PRIO_MAX}
8830		{OPEN_MAX}
8831		A value one greater than the maximum value that the system may assign to a newly-created file descriptor.
8832		Minimum Acceptable Value: {_POSIX_OPEN_MAX}
8833		
8834		{PAGESIZE}
8835		Size in bytes of a page.
8836		Minimum Acceptable Value: 1

<limits.h>

Headers

8837	XSI	{PAGE_SIZE}
8838		Equivalent to {PAGESIZE}. If either {PAGESIZE} or {PAGE_SIZE} is defined, the other is
8839		defined with the same value.
8840		{PTHREAD_DESTRUCTOR_ITERATIONS}
8841		Maximum number of attempts made to destroy a thread's thread-specific data values on
8842		thread exit.
8843		Minimum Acceptable Value: {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
8844		{PTHREAD_KEYS_MAX}
8845		Maximum number of data keys that can be created by a process.
8846		Minimum Acceptable Value: {_POSIX_THREAD_KEYS_MAX}
8847		{PTHREAD_STACK_MIN}
8848		Minimum size in bytes of thread stack storage.
8849		Minimum Acceptable Value: 0
8850		{PTHREAD_THREADS_MAX}
8851		Maximum number of threads that can be created per process.
8852		Minimum Acceptable Value: {_POSIX_THREAD_THREADS_MAX}
8853		{RE_DUP_MAX}
8854		Maximum number of repeated occurrences of a BRE or ERE interval expression; see Section
8855		9.3.6 (on page 186) and Section 9.4.6 (on page 189).
8856		Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}
8857		{RTSIG_MAX}
8858		Maximum number of realtime signals reserved for application use in this implementation.
8859		Minimum Acceptable Value: {_POSIX_RTSIG_MAX}
8860		{SEM_NSEMS_MAX}
8861		Maximum number of semaphores that a process may have.
8862		Minimum Acceptable Value: {_POSIX_SEM_NSEMS_MAX}
8863		{SEM_VALUE_MAX}
8864		The maximum value a semaphore may have.
8865		Minimum Acceptable Value: {_POSIX_SEM_VALUE_MAX}
8866		{SIGQUEUE_MAX}
8867		Maximum number of queued signals that a process may send and have pending at the
8868		receiver(s) at any time.
8869		Minimum Acceptable Value: {_POSIX_SIGQUEUE_MAX}
8870	SS TSP	{SS_REPL_MAX}
8871		The maximum number of replenishment operations that may be simultaneously pending
8872		for a particular sporadic server scheduler.
8873		Minimum Acceptable Value: {_POSIX_SS_REPL_MAX}
8874		{STREAM_MAX}
8875		Maximum number of streams that one process can have open at one time. If defined, it has
8876		the same value as {FOPEN_MAX} (see <stdio.h>).
8877		Minimum Acceptable Value: {_POSIX_STREAM_MAX}
8878		{SYMLOOP_MAX}
8879		Maximum number of symbolic links that can be reliably traversed in the resolution of a
8880		pathname in the absence of a loop.
8881		Minimum Acceptable Value: {_POSIX_SYMLOOP_MAX}

Headers

<limits.h>

8882 {TIMER_MAX}
 8883 Maximum number of timers per process supported by the implementation.
 8884 Minimum Acceptable Value: {_POSIX_TIMER_MAX}

8885 OB TRC {TRACE_EVENT_NAME_MAX}
 8886 Maximum length of the trace event name (not including the terminating null).
 8887 Minimum Acceptable Value: {_POSIX_TRACE_EVENT_NAME_MAX}

8888 OB TRC {TRACE_NAME_MAX}
 8889 Maximum length of the trace generation version string or of the trace stream name (not
 8890 including the terminating null).
 8891 Minimum Acceptable Value: {_POSIX_TRACE_NAME_MAX}

8892 OB TRC {TRACE_SYS_MAX}
 8893 Maximum number of trace streams that may simultaneously exist in the system.
 8894 Minimum Acceptable Value: {_POSIX_TRACE_SYS_MAX}

8895 OB TRC {TRACE_USER_EVENT_MAX}
 8896 Maximum number of user trace event type identifiers that may simultaneously exist in a
 8897 traced process, including the predefined user trace event
 8898 POSIX_TRACE_UNNAMED_USER_EVENT.
 8899 Minimum Acceptable Value: {_POSIX_TRACE_USER_EVENT_MAX}

8900 {TTY_NAME_MAX}
 8901 Maximum length of terminal device name.
 8902 Minimum Acceptable Value: {_POSIX_TTY_NAME_MAX}

8903 {TZNAME_MAX}
 8904 Maximum number of bytes supported for the name of a timezone (not of the TZ variable).
 8905 Minimum Acceptable Value: {_POSIX_TZNAME_MAX}

8906 **Note:** The length given by {TZNAME_MAX} does not include the quoting characters mentioned in
 8907 [Section 8.3](#) (on page 177).

Pathname Variable Values

8908
 8909 The values in the following list may be constants within an implementation or may vary from
 8910 one pathname to another. For example, file systems or directories may have different
 8911 characteristics.

8912 A definition of one of the symbolic constants in the following list shall be omitted from the
 8913 <limits.h> header on specific implementations where the corresponding value is equal to or
 8914 greater than the stated minimum, but where the value can vary depending on the file to which it
 8915 is applied. The actual value supported for a specific pathname shall be provided by the
 8916 `pathconf()` function.

8917 {FILESIZEBITS}
 8918 Minimum number of bits needed to represent, as a signed integer value, the maximum size
 8919 of a regular file allowed in the specified directory.
 8920 Minimum Acceptable Value: 32

8921 {LINK_MAX}
 8922 Maximum number of links to a single file.
 8923 Minimum Acceptable Value: {_POSIX_LINK_MAX}

8924 {MAX_CANON}
 8925 Maximum number of bytes in a terminal canonical input line.
 8926 Minimum Acceptable Value: {_POSIX_MAX_CANON}

<limits.h>

Headers

8927		{MAX_INPUT}
8928		Minimum number of bytes for which space is available in a terminal input queue; therefore,
8929		the maximum number of bytes a conforming application may require to be typed as input
8930		before reading them.
8931		Minimum Acceptable Value: {_POSIX_MAX_INPUT}
8932		{NAME_MAX}
8933		Maximum number of bytes in a filename (not including the terminating null).
8934		Minimum Acceptable Value: {_POSIX_NAME_MAX}
8935	XSI	Minimum Acceptable Value: {_XOPEN_NAME_MAX}
8936		{PATH_MAX}
8937		Maximum number of bytes the implementation will store as a pathname in a user-supplied
8938		buffer of unspecified size, including the terminating null character. Minimum number the
8939		implementation will accept as the maximum number of bytes in a pathname.
8940		Minimum Acceptable Value: {_POSIX_PATH_MAX}
8941	XSI	Minimum Acceptable Value: {_XOPEN_PATH_MAX}
8942		{PIPE_BUF}
8943		Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
8944		Minimum Acceptable Value: {_POSIX_PIPE_BUF}
8945	ADV	{POSIX_ALLOC_SIZE_MIN}
8946		Minimum number of bytes of storage actually allocated for any portion of a file.
8947		Minimum Acceptable Value: Not specified.
8948	ADV	{POSIX_REC_INCR_XFER_SIZE}
8949		Recommended increment for file transfer sizes between the
8950		{POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.
8951		Minimum Acceptable Value: Not specified.
8952	ADV	{POSIX_REC_MAX_XFER_SIZE}
8953		Maximum recommended file transfer size.
8954		Minimum Acceptable Value: Not specified.
8955	ADV	{POSIX_REC_MIN_XFER_SIZE}
8956		Minimum recommended file transfer size.
8957		Minimum Acceptable Value: Not specified.
8958	ADV	{POSIX_REC_XFER_ALIGN}
8959		Recommended file transfer buffer alignment.
8960		Minimum Acceptable Value: Not specified.
8961		{SYMLINK_MAX}
8962		Maximum number of bytes in a symbolic link.
8963		Minimum Acceptable Value: {_POSIX_SYMLINK_MAX}

Runtime Increaseable Values

8965 The magnitude limitations in the following list shall be fixed by specific implementations. An
 8966 application should assume that the value of the symbolic constant defined by <limits.h> in a
 8967 specific implementation is the minimum that pertains whenever the application is run under
 8968 that implementation. A specific instance of a specific implementation may increase the value
 8969 relative to that supplied by <limits.h> for that implementation. The actual value supported by a
 8970 specific instance shall be provided by the *sysconf()* function.

Headers

<limits.h>

8971	{BC_BASE_MAX}	
8972	Maximum <i>obase</i> values allowed by the <i>bc</i> utility.	
8973	Minimum Acceptable Value: {_POSIX2_BC_BASE_MAX}	
8974	{BC_DIM_MAX}	
8975	Maximum number of elements permitted in an array by the <i>bc</i> utility.	
8976	Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}	
8977	{BC_SCALE_MAX}	
8978	Maximum <i>scale</i> value allowed by the <i>bc</i> utility.	
8979	Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}	
8980	{BC_STRING_MAX}	
8981	Maximum length of a string constant accepted by the <i>bc</i> utility.	
8982	Minimum Acceptable Value: {_POSIX2_BC_STRING_MAX}	
8983	{CHARCLASS_NAME_MAX}	
8984	Maximum number of bytes in a character class name.	
8985	Minimum Acceptable Value: {_POSIX2_CHARCLASS_NAME_MAX}	
8986	{COLL_WEIGHTS_MAX}	
8987	Maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> order	
8988	keyword in the locale definition file; see Chapter 7 (on page 135).	
8989	Minimum Acceptable Value: {_POSIX2_COLL_WEIGHTS_MAX}	
8990	{EXPR_NEST_MAX}	
8991	Maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	
8992	Minimum Acceptable Value: {_POSIX2_EXPR_NEST_MAX}	
8993	{LINE_MAX}	
8994	Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either	
8995	standard input or another file) when the utility is described as processing text files. The	
8996	length includes room for the trailing <newline>.	
8997	Minimum Acceptable Value: {_POSIX2_LINE_MAX}	
8998	{NGROUPS_MAX}	
8999	Maximum number of simultaneous supplementary group IDs per process.	
9000	Minimum Acceptable Value: {_POSIX2_NGROUPS_MAX}	
9001	{RE_DUP_MAX}	
9002	Maximum number of repeated occurrences of a regular expression permitted when using	
9003	the interval notation $\{m,n\}$; see Chapter 9 (on page 181).	
9004	Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}	
9005	Maximum Values	
9006	The <limits.h> header shall define the following symbolic constants with the values shown.	
9007	These are the most restrictive values for certain features on an implementation. A conforming	
9008	implementation shall provide values no larger than these values. A conforming application must	
9009	not require a smaller value for correct operation.	
9010	{_POSIX_CLOCKRES_MIN}	
9011	The resolution of the CLOCK_REALTIME clock, in nanoseconds.	
9012	Value: 20 000 000	
9013	MON	If the Monotonic Clock option is supported, the resolution of the CLOCK_MONOTONIC
9014		clock, in nanoseconds, is represented by {_POSIX_CLOCKRES_MIN}.

<limits.h>

Headers

9015 Minimum Values

9016 The <limits.h> header shall define the following symbolic constants with the values shown.
 9017 These are the most restrictive values for certain features on an implementation conforming to
 9018 this volume of POSIX.1-2008. Related symbolic constants are defined elsewhere in this volume
 9019 of POSIX.1-2008 which reflect the actual implementation and which need not be as restrictive. A
 9020 conforming implementation shall provide values at least this large. A strictly conforming
 9021 application must not require a larger value for correct operation.

9022 {_POSIX_AIO_LISTIO_MAX}

9023 The number of I/O operations that can be specified in a list I/O call.

9024 Value: 2

9025 {_POSIX_AIO_MAX}

9026 The number of outstanding asynchronous I/O operations.

9027 Value: 1

9028 {_POSIX_ARG_MAX}

9029 Maximum length of argument to the *exec* functions including environment data.

9030 Value: 4 096

9031 {_POSIX_CHILD_MAX}

9032 Maximum number of simultaneous processes per real user ID.

9033 Value: 25

9034 {_POSIX_DELAYTIMER_MAX}

9035 The number of timer expiration overruns.

9036 Value: 32

9037 {_POSIX_HOST_NAME_MAX}

9038 Maximum length of a host name (not including the terminating null) as returned from the
 9039 *gethostname()* function.

9040 Value: 255

9041 {_POSIX_LINK_MAX}

9042 Maximum number of links to a single file.

9043 Value: 8

9044 {_POSIX_LOGIN_NAME_MAX}

9045 The size of the storage required for a login name, in bytes (including the terminating null).

9046 Value: 9

9047 {_POSIX_MAX_CANON}

9048 Maximum number of bytes in a terminal canonical input queue.

9049 Value: 255

9050 {_POSIX_MAX_INPUT}

9051 Maximum number of bytes allowed in a terminal input queue.

9052 Value: 255

9053 MSG {_POSIX_MQ_OPEN_MAX}

9054 The number of message queues that can be open for a single process.

9055 Value: 8

9056 MSG {_POSIX_MQ_PRIO_MAX}

9057 The maximum number of message priorities supported by the implementation.

9058 Value: 32

Headers

<limits.h>

9059 `{_POSIX_NAME_MAX}`
 9060 Maximum number of bytes in a filename (not including the terminating null).
 9061 Value: 14

9062 `{_POSIX_NGROUPS_MAX}`
 9063 Maximum number of simultaneous supplementary group IDs per process.
 9064 Value: 8

9065 `{_POSIX_OPEN_MAX}`
 9066 Maximum number of files that one process can have open at any one time.
 9067 Value: 20

9068 `{_POSIX_PATH_MAX}`
 9069 Minimum number the implementation will accept as the maximum number of bytes in a
 9070 pathname.
 9071 Value: 256

9072 `{_POSIX_PIPE_BUF}`
 9073 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
 9074 Value: 512

9075 `{_POSIX_RE_DUP_MAX}`
 9076 The number of repeated occurrences of a BRE permitted by the *regex*(*re*) and *regcomp*(*re*)
 9077 functions when using the interval notation *{m,n}*; see Section 9.3.6 (on page 186).
 9078 Value: 255

9079 `{_POSIX_RTSIG_MAX}`
 9080 The number of realtime signal numbers reserved for application use.
 9081 Value: 8

9082 `{_POSIX_SEM_NSEMS_MAX}`
 9083 The number of semaphores that a process may have.
 9084 Value: 256

9085 `{_POSIX_SEM_VALUE_MAX}`
 9086 The maximum value a semaphore may have.
 9087 Value: 32 767

9088 `{_POSIX_SIGQUEUE_MAX}`
 9089 The number of queued signals that a process may send and have pending at the receiver(s)
 9090 at any time.
 9091 Value: 32

9092 `{_POSIX_SSIZE_MAX}`
 9093 The value that can be stored in an object of type *ssize_t*.
 9094 Value: 32 767

9095 SS | TSP `{_POSIX_SS_REPL_MAX}`
 9096 The number of replenishment operations that may be simultaneously pending for a
 9097 particular sporadic server scheduler.
 9098 Value: 4

9099 `{_POSIX_STREAM_MAX}`
 9100 The number of streams that one process can have open at one time.
 9101 Value: 8

<limits.h>

Headers

9102		{_POSIX_SYMLINK_MAX}
9103		The number of bytes in a symbolic link.
9104		Value: 255
9105		{_POSIX_SYMLOOP_MAX}
9106		The number of symbolic links that can be traversed in the resolution of a pathname in the
9107		absence of a loop.
9108		Value: 8
9109		{_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
9110		The number of attempts made to destroy a thread's thread-specific data values on thread
9111		exit.
9112		Value: 4
9113		{_POSIX_THREAD_KEYS_MAX}
9114		The number of data keys per process.
9115		Value: 128
9116		{_POSIX_THREAD_THREADS_MAX}
9117		The number of threads per process.
9118		Value: 64
9119		{_POSIX_TIMER_MAX}
9120		The per-process number of timers.
9121		Value: 32
9122	OB TRC	{_POSIX_TRACE_EVENT_NAME_MAX}
9123		The length in bytes of a trace event name (not including the terminating null).
9124		Value: 30
9125	OB TRC	{_POSIX_TRACE_NAME_MAX}
9126		The length in bytes of a trace generation version string or a trace stream name (not
9127		including the terminating null).
9128		Value: 8
9129	OB TRC	{_POSIX_TRACE_SYS_MAX}
9130		The number of trace streams that may simultaneously exist in the system.
9131		Value: 8
9132	OB TRC	{_POSIX_TRACE_USER_EVENT_MAX}
9133		The number of user trace event type identifiers that may simultaneously exist in a traced
9134		process, including the predefined user trace event
9135		POSIX_TRACE_UNNAMED_USER_EVENT.
9136		Value: 32
9137		{_POSIX_TTY_NAME_MAX}
9138		The size of the storage required for a terminal device name, in bytes (including the
9139		terminating null).
9140		Value: 9
9141		{_POSIX_TZNAME_MAX}
9142		Maximum number of bytes supported for the name of a timezone (not of the TZ variable).
9143		Value: 6
9144	Note:	The length given by {_POSIX_TZNAME_MAX} does not include the quoting characters
9145		mentioned in Section 8.3 (on page 177).

Headers

<limits.h>

9146		<code>{_POSIX2_BC_BASE_MAX}</code>
9147		Maximum <i>obase</i> values allowed by the <i>bc</i> utility.
9148		Value: 99
9149		<code>{_POSIX2_BC_DIM_MAX}</code>
9150		Maximum number of elements permitted in an array by the <i>bc</i> utility.
9151		Value: 2 048
9152		<code>{_POSIX2_BC_SCALE_MAX}</code>
9153		Maximum <i>scale</i> value allowed by the <i>bc</i> utility.
9154		Value: 99
9155		<code>{_POSIX2_BC_STRING_MAX}</code>
9156		Maximum length of a string constant accepted by the <i>bc</i> utility.
9157		Value: 1 000
9158		<code>{_POSIX2_CHARCLASS_NAME_MAX}</code>
9159		Maximum number of bytes in a character class name.
9160		Value: 14
9161		<code>{_POSIX2_COLL_WEIGHTS_MAX}</code>
9162		Maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> order
9163		keyword in the locale definition file; see Chapter 7 (on page 135).
9164		Value: 2
9165		<code>{_POSIX2_EXPR_NEST_MAX}</code>
9166		Maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.
9167		Value: 32
9168		<code>{_POSIX2_LINE_MAX}</code>
9169		Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
9170		standard input or another file) when the utility is described as processing text files. The
9171		length includes room for the trailing <newline>.
9172		Value: 2 048
9173		<code>{_POSIX2_RE_DUP_MAX}</code>
9174		Maximum number of repeated occurrences of a regular expression permitted when using
9175		the interval notation <code>\{m,n\}</code> ; see Chapter 9 (on page 181).
9176		Value: 255
9177	XSI	<code>{_XOPEN_IOV_MAX}</code>
9178		Maximum number of iovec structures that one process has available for use with <i>readv()</i> or
9179		<i>writev()</i> .
9180		Value: 16
9181	XSI	<code>{_XOPEN_NAME_MAX}</code>
9182		Maximum number of bytes in a filename (not including the terminating null).
9183		Value: 255
9184	XSI	<code>{_XOPEN_PATH_MAX}</code>
9185		Minimum number the implementation will accept as the maximum number of bytes in a
9186		pathname.
9187		Value: 1 024

<limits.h>

Headers

9188 Numerical Limits

9189 The <limits.h> header shall define the following macros and, except for {CHAR_BIT},
 9190 {LONG_BIT}, {MB_LEN_MAX}, and {WORD_BIT}, they shall be replaced by expressions that
 9191 have the same type as would an expression that is an object of the corresponding type converted
 9192 according to the integer promotions.

9193 If the value of an object of type **char** is treated as a signed integer when used in an expression,
 9194 the value of {CHAR_MIN} is the same as that of {SCHAR_MIN} and the value of {CHAR_MAX}
 9195 is the same as that of {SCHAR_MAX}. Otherwise, the value of {CHAR_MIN} is 0 and the value
 9196 of {CHAR_MAX} is the same as that of {UCHAR_MAX}.

9197 {CHAR_BIT}

9198 Number of bits in a type **char**.

9199 CX Value: 8

9200 {CHAR_MAX}

9201 Maximum value for an object of type **char**.

9202 Value: {UCHAR_MAX} or {SCHAR_MAX}

9203 {CHAR_MIN}

9204 Minimum value for an object of type **char**.

9205 Value: {SCHAR_MIN} or 0

9206 {INT_MAX}

9207 Maximum value for an object of type **int**.

9208 CX Minimum Acceptable Value: 2 147 483 647

9209 {INT_MIN}

9210 Minimum value for an object of type **int**.

9211 CX Maximum Acceptable Value: -2 147 483 647

9212 {LLONG_MAX}

9213 Maximum value for an object of type **long long**.

9214 Minimum Acceptable Value: +9 223 372 036 854 775 807

9215 {LLONG_MIN}

9216 Minimum value for an object of type **long long**.

9217 Maximum Acceptable Value: -9 223 372 036 854 775 807

9218 {LONG_BIT}

9219 Number of bits in an object of type **long**.

9220 Minimum Acceptable Value: 32

9221 {LONG_MAX}

9222 Maximum value for an object of type **long**.

9223 Minimum Acceptable Value: +2 147 483 647

9224 {LONG_MIN}

9225 Minimum value for an object of type **long**.

9226 Maximum Acceptable Value: -2 147 483 647

9227 {MB_LEN_MAX}

9228 Maximum number of bytes in a character, for any supported locale.

9229 Minimum Acceptable Value: 1

9230 {SCHAR_MAX}

9231 Maximum value for an object of type **signed char**.

Headers

<limits.h>

9232	CX	Value: +127
9233		{SCHAR_MIN}
9234		Minimum value for an object of type signed char .
9235	CX	Value: -128
9236		{SHRT_MAX}
9237		Maximum value for an object of type short .
9238		Minimum Acceptable Value: +32 767
9239		{SHRT_MIN}
9240		Minimum value for an object of type short .
9241		Maximum Acceptable Value: -32 767
9242		{SSIZE_MAX}
9243		Maximum value for an object of type ssize_t .
9244		Minimum Acceptable Value: {_POSIX_SSIZE_MAX}
9245		{UCHAR_MAX}
9246		Maximum value for an object of type unsigned char .
9247	CX	Value: 255
9248		{UINT_MAX}
9249		Maximum value for an object of type unsigned .
9250	CX	Minimum Acceptable Value: 4 294 967 295
9251		{ULLONG_MAX}
9252		Maximum value for an object of type unsigned long long .
9253		Minimum Acceptable Value: 18 446 744 073 709 551 615
9254		{ULONG_MAX}
9255		Maximum value for an object of type unsigned long .
9256		Minimum Acceptable Value: 4 294 967 295
9257		{USHRT_MAX}
9258		Maximum value for an object of type unsigned short .
9259		Minimum Acceptable Value: 65 535
9260		{WORD_BIT}
9261		Number of bits in an object of type int .
9262		Minimum Acceptable Value: 32
9263		Other Invariant Values
9264		The <limits.h> header shall define the following symbolic constants:
9265		{NL_ARGMAX}
9266		Maximum value of <i>n</i> in conversion specifications using the "%n\$" sequence in calls to the <i>printf()</i> and <i>scanf()</i> families of functions.
9267		
9268		Minimum Acceptable Value: 9
9269	XSI	{NL_LANGMAX}
9270		Maximum number of bytes in a <i>LANG</i> name.
9271		Minimum Acceptable Value: 14
9272		{NL_MSGMAX}
9273		Maximum message number.
9274		Minimum Acceptable Value: 32 767

<limits.h>

Headers

9275 {NL_SETMAX}
 9276 Maximum set number.
 9277 Minimum Acceptable Value: 255

9278 {NL_TEXTMAX}
 9279 Maximum number of bytes in a message string.
 9280 Minimum Acceptable Value: {_POSIX2_LINE_MAX}

9281 XSI {NZERO}
 9282 Default process priority.
 9283 Minimum Acceptable Value: 20

9284 APPLICATION USAGE

9285 None.

9286 RATIONALE

9287 A request was made to reduce the value of {_POSIX_LINK_MAX} from the value of 8 specified
 9288 for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request
 9289 for several reasons:

- 9290 • They wanted to avoid making any changes to the standard that could break conforming
 9291 applications, and the requested change could have that effect.
- 9292 • The use of multiple hard links to a file cannot always be replaced with use of symbolic
 9293 links. Symbolic links are semantically different from hard links in that they associate a
 9294 pathname with another pathname rather than a pathname with a file. This has
 9295 implications for access control, file permanence, and transparency.
- 9296 • The original standard developers had considered the issue of allowing for
 9297 implementations that did not in general support hard links, and decided that this would
 9298 reduce consensus on the standard.

9299 Systems that support historical versions of the development option of the ISO POSIX-2 standard
 9300 retain the name {_POSIX2_RE_DUP_MAX} as an alias for {_POSIX_RE_DUP_MAX}.

9301 {PATH_MAX}
 9302 IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the
 9303 definition of pathname and the description of {PATH_MAX}, allowing application
 9304 developers to allocate either {PATH_MAX} or {PATH_MAX}+1 bytes. The inconsistency has
 9305 been removed by correction to the {PATH_MAX} definition to include the null character.
 9306 With this change, applications that previously allocated {PATH_MAX} bytes will continue to
 9307 succeed.

9308 {SYMLINK_MAX}
 9309 This symbol refers to space for data that is stored in the file system, as opposed to
 9310 {PATH_MAX} which is the length of a name that can be passed to a function. In some
 9311 existing implementations, the filenames pointed to by symbolic links are stored in the *inodes*
 9312 of the links, so it is important that {SYMLINK_MAX} not be constrained to be as large as
 9313 {PATH_MAX}.

9314 FUTURE DIRECTIONS

9315 None.

9316 SEE ALSO

9317 Chapter 7 (on page 135), <stdio.h>, <unistd.h>

9318 XSH Section 2.2 (on page 468), *fpathconf()*, *sysconf()*

CHANGE HISTORY

First released in Issue 1.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

{FILESIZEBITS} is added for the Large File Summit extensions.

The minimum acceptable values for {INT_MAX}, {INT_MIN}, and {UINT_MAX} are changed to make 32-bit values the minimum requirement.

The entry is restructured to improve readability.

Issue 6

The Open Group Corrigendum U033/4 is applied. The wording is made clear for {CHAR_MIN}, {INT_MIN}, {LONG_MIN}, {SCHAR_MIN}, and {SHRT_MIN} that these are maximum acceptable values.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The minimum value for {CHILD_MAX} is 25. This is a FIPS requirement.
- The minimum value for {OPEN_MAX} is 20. This is a FIPS requirement.
- The minimum value for {NGROUPS_MAX} is 8. This is also a FIPS requirement.

Symbolic constants are added for {_POSIX_SYMLINK_MAX}, {_POSIX_SYMLOOP_MAX}, {_POSIX_RE_DUP_MAX}, {RE_DUP_MAX}, {SYMLOOP_MAX}, and {SYMLINK_MAX}.

The following values are added for alignment with IEEE Std 1003.1d-1999:

{_POSIX_SS_REPL_MAX}
 {SS_REPL_MAX}
 {POSIX_ALLOC_SIZE_MIN}
 {POSIX_REC_INCR_XFER_SIZE}
 {POSIX_REC_MAX_XFER_SIZE}
 {POSIX_REC_MIN_XFER_SIZE}
 {POSIX_REC_XFER_ALIGN}

Reference to CLOCK_MONOTONIC is added in the description of {_POSIX_CLOCKRES_MIN} for alignment with IEEE Std 1003.1j-2000.

The constants {LLONG_MIN}, {LLONG_MAX}, and {ULLONG_MAX} are added for alignment with the ISO/IEC 9899:1999 standard.

The following values are added for alignment with IEEE Std 1003.1q-2000:

{_POSIX_TRACE_EVENT_NAME_MAX}
 {_POSIX_TRACE_NAME_MAX}
 {_POSIX_TRACE_SYS_MAX}
 {_POSIX_TRACE_USER_EVENT_MAX}
 {TRACE_EVENT_NAME_MAX}
 {TRACE_NAME_MAX}
 {TRACE_SYS_MAX}
 {TRACE_USER_EVENT_MAX}

The new limits {_XOPEN_NAME_MAX} and {_XOPEN_PATH_MAX} are added as minimum

<limits.h>

Headers

- 9361 values for {PATH_MAX} and {NAME_MAX} limits on XSI-conformant systems.
- 9362 The LEGACY symbols {PASS_MAX} and {TMP_MAX} are removed.
- 9363 The values for the limits {CHAR_BIT}, {SCHAR_MAX}, and {UCHAR_MAX} are now required
9364 to be 8, +127, and 255, respectively.
- 9365 The value for the limit {CHAR_MAX} is now {UCHAR_MAX} or {SCHAR_MAX}.
- 9366 The value for the limit {CHAR_MIN} is now {SCHAR_MIN} or zero.
- 9367 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/10 is applied, correcting the value of
9368 {_POSIX_CHILD_MAX} from 6 to 25. This is for FIPS 151-2 alignment.
- 9369 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/19 is applied, updating the values for
9370 {INT_MAX}, {UINT_MAX}, and {INT_MIN} to be CX extensions over the ISO C standard, and
9371 correcting {WORD_BIT} from 16 to 32.
- 9372 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/20 is applied, removing
9373 {CHARCLASS_NAME_MAX} from the “Other Invariant Values” section (it also occurs under
9374 “Runtime Increaseable Values”).
- 9375 **Issue 7**
- 9376 Austin Group Interpretations 1003.1-2001 #143 and #160 are applied.
- 9377 Austin Group Interpretation 1003.1-2001 #173 is applied, updating the descriptions of
9378 {TRACE_EVENT_NAME_MAX} and {TRACE_NAME_MAX} to not include the terminating
9379 null.
- 9380 SD5-XBD-ERN-36 is applied, changing the description of {RE_DUP_MAX}.
- 9381 SD5-XBD-ERN-90 is applied.
- 9382 {NL_NMAX} is removed; it should have been removed in Issue 6.
- 9383 The Trace option values are marked obsolescent.
- 9384 The {ATEXIT_MAX}, {LONG_BIT}, {NL_MSGMAX}, {NL_SETMAX}, {NL_TEXTMAX}, and
9385 {WORD_BIT} values are moved from the XSI option to the Base.
- 9386 The AIO_* and _POSIX_AIO_* values are moved from the Asynchronous Input and Output
9387 option to the Base.
- 9388 The {_POSIX_RTSIG_MAX}, {_POSIX_SIGQUEUE_MAX}, {RTSIG_MAX}, and
9389 {SIGQUEUE_MAX} values are moved from the Realtime Signals Extension option to the Base.
- 9390 Functionality relating to the Threads and Timers options is moved to the Base.
- 9391 This reference page is clarified with respect to macros and symbolic constants.

Headers

<locale.h>

NAME

locale.h — category macros

SYNOPSIS

```
#include <locale.h>
```

DESCRIPTION

Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to enable the visibility of these symbols in this header.

The <locale.h> header shall define the **lconv** structure, which shall include at least the following members. (See the definitions of *LC_MONETARY* in Section 7.3.3 (on page 154) and Section 7.3.4 (on page 157).)

```
char    *currency_symbol
char    *decimal_point
char    frac_digits
char    *grouping
char    *int_curr_symbol
char    int_frac_digits
char    int_n_cs_precedes
char    int_n_sep_by_space
char    int_n_sign_posn
char    int_p_cs_precedes
char    int_p_sep_by_space
char    int_p_sign_posn
char    *mon_decimal_point
char    *mon_grouping
char    *mon_thousands_sep
char    *negative_sign
char    n_cs_precedes
char    n_sep_by_space
char    n_sign_posn
char    *positive_sign
char    p_cs_precedes
char    p_sep_by_space
char    p_sign_posn
char    *thousands_sep
```

The <locale.h> header shall define NULL (as described in <stddef.h>) and at least the following as macros:

```
LC_ALL
LC_COLLATE
LC_CTYPE
LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME
```

which shall expand to integer constant expressions with distinct values for use as the first argument to the *setlocale()* function.

Implementations may add additional masks using the form *LC_** and an uppercase letter.

<locale.h>*Headers*

9439 CX The **<locale.h>** header shall contain at least the following macros representing bitmasks for use
 9440 with the *newlocale()* function for each supported locale category:

9441 LC_COLLATE_MASK
 9442 LC_CTYPE_MASK
 9443 LC_MESSAGES_MASK
 9444 LC_MONETARY_MASK
 9445 LC_NUMERIC_MASK
 9446 LC_TIME_MASK

9447 Implementations may add additional masks using the form LC_*_MASK.

9448 In addition, a macro to set the bits for all categories set shall be defined:

9449 LC_ALL_MASK

9450 The **<locale.h>** header shall define LC_GLOBAL_LOCALE, a special locale object descriptor
 9451 used by the *uselocale()* function.

9452 The **<locale.h>** header shall define the **locale_t** type, representing a locale object.

9453 The following shall be declared as functions and may also be defined as macros. Function
 9454 prototypes shall be provided for use with ISO C standard compilers.

9455 CX locale_t duplocale(locale_t);
 9456 void freelocale(locale_t);
 9457 struct lconv *localeconv(void);

9458 CX locale_t newlocale(int, const char *, locale_t);
 9459 char *setlocale(int, const char *);

9460 CX locale_t uselocale (locale_t);

9461 **APPLICATION USAGE**

9462 None.

9463 **RATIONALE**

9464 None.

9465 **FUTURE DIRECTIONS**

9466 None.

9467 **SEE ALSO**9468 Chapter 8 (on page 173), **<stddef.h>**9469 XSH *duplocale()*, *freelocale()*, *localeconv()*, *newlocale()*, *setlocale()*, *uselocale()*9470 **CHANGE HISTORY**

9471 First released in Issue 3.

9472 Included for alignment with the ISO C standard.

9473 **Issue 6**

9474 The **lconv** structure is expanded with new members (**int_n_cs_precedes**, **int_n_sep_by_space**,
 9475 **int_n_sign_posn**, **int_p_cs_precedes**, **int_p_sep_by_space**, and **int_p_sign_posn**) for alignment
 9476 with the ISO/IEC 9899:1999 standard.

9477 Extensions beyond the ISO C standard are marked.

*Headers***<locale.h>**9478 **Issue 7**

9479 The *duplocale()*, *freelocale()*, *newlocale()*, and *uselocale()* functions are added from The Open
9480 Group Technical Standard, 2006, Extended API Set Part 4.

9481 This reference page is clarified with respect to macros and symbolic constants.

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

<math.h>

Headers

9482 NAME

9483 math.h — mathematical declarations

9484 SYNOPSIS

9485 #include <math.h>

9486 DESCRIPTION

9487 CX Some of the functionality described on this reference page extends the ISO C standard.
 9488 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to
 9489 enable the visibility of these symbols in this header.

9490 The <math.h> header shall define at least the following types:

9491 **float_t** A real-floating type at least as wide as **float**.9492 **double_t** A real-floating type at least as wide as **double**, and at least as wide as **float_t**.

9493 If FLT_EVAL_METHOD equals 0, **float_t** and **double_t** shall be **float** and **double**, respectively; if
 9494 FLT_EVAL_METHOD equals 1, they shall both be **double**; if FLT_EVAL_METHOD equals 2,
 9495 they shall both be **long double**; for other values of FLT_EVAL_METHOD, they are otherwise
 9496 implementation-defined.

9497 The <math.h> header shall define the following macros, where real-floating indicates that the
 9498 argument shall be an expression of real-floating type:

```
9499 int fpclassify(real-floating x);
9500 int isfinite(real-floating x);
9501 int isgreater(real-floating x, real-floating y);
9502 int isgreaterequal(real-floating x, real-floating y);
9503 int isinf(real-floating x);
9504 int isless(real-floating x, real-floating y);
9505 int islessequal(real-floating x, real-floating y);
9506 int islessgreater(real-floating x, real-floating y);
9507 int isnan(real-floating x);
9508 int isnormal(real-floating x);
9509 int isunordered(real-floating x, real-floating y);
9510 int signbit(real-floating x);
```

9511 The <math.h> header shall define the following symbolic constants. The values shall have type
 9512 **double** and shall be accurate within the precision of the **double** type.

9513	XSI	M_E	Value of e
9514	XSI	M_LOG2E	Value of $\log_2 e$
9515	XSI	M_LOG10E	Value of $\log_{10} e$
9516	XSI	M_LN2	Value of $\log_e 2$
9517	XSI	M_LN10	Value of $\log_e 10$
9518	XSI	M_PI	Value of π
9519	XSI	M_PI_2	Value of $\pi/2$
9520	XSI	M_PI_4	Value of $\pi/4$
9521	XSI	M_1_PI	Value of $1/\pi$
9522	XSI	M_2_PI	Value of $2/\pi$

Headers

<math.h>

9523 XSI **M_2_SQRTPI** Value of $2/\sqrt{\pi}$

9524 XSI **M_SQRT2** Value of $\sqrt{2}$

9525 XSI **M_SQRT1_2** Value of $1/\sqrt{2}$

9526 The <math.h> header shall define the following symbolic constant:

9527 OB XSI **MAXFLOAT** Same value as **FLT_MAX** in <float.h>.

9528 The <math.h> header shall define the following macros:

9529 **HUGE_VAL** A positive **double** constant expression, not necessarily representable as a
9530 **float**. Used as an error value returned by the mathematics library.
9531 **HUGE_VAL** evaluates to +infinity on systems supporting IEEE Std 754-1985.

9532 **HUGE_VALF** A positive **float** constant expression. Used as an error value returned by the
9533 mathematics library. **HUGE_VALF** evaluates to +infinity on systems
9534 supporting IEEE Std 754-1985.

9535 **HUGE_VALL** A positive **long double** constant expression. Used as an error value returned
9536 by the mathematics library. **HUGE_VALL** evaluates to +infinity on systems
9537 supporting IEEE Std 754-1985.

9538 **INFINITY** A constant expression of type **float** representing positive or unsigned infinity,
9539 if available; else a positive constant of type **float** that overflows at translation
9540 time.

9541 **NAN** A constant expression of type **float** representing a quiet NaN. This macro is
9542 only defined if the implementation supports quiet NaNs for the **float** type.

9543 The following macros shall be defined for number classification. They represent the mutually-
9544 exclusive kinds of floating-point values. They expand to integer constant expressions with
9545 distinct values. Additional implementation-defined floating-point classifications, with macro
9546 definitions beginning with **FP_** and an uppercase letter, may also be specified by the
9547 implementation.

9548 **FP_INFINITE**
9549 **FP_NAN**
9550 **FP_NORMAL**
9551 **FP_SUBNORMAL**
9552 **FP_ZERO**

9553 The following optional macros indicate whether the *fma()* family of functions are fast compared
9554 with direct code:

9555 **FP_FAST_FMA**
9556 **FP_FAST_FMAF**
9557 **FP_FAST_FMAL**

9558 If defined, the **FP_FAST_FMA** macro shall expand to the integer constant 1 and shall indicate
9559 that the *fma()* function generally executes about as fast as, or faster than, a multiply and an add
9560 of **double** operands. If undefined, the speed of execution is unspecified. The other macros have
9561 the equivalent meaning for the **float** and **long double** versions.

9562 The following macros shall expand to integer constant expressions whose values are returned by
9563 *ilogb(x)* if *x* is zero or NaN, respectively. The value of **FP_ILOGB0** shall be either {INT_MIN} or
9564 -{INT_MAX}. The value of **FP_ILOGBNAN** shall be either {INT_MAX} or {INT_MIN}.

<math.h>

Headers

9565 FP_ILOGB0
 9566 FP_ILOGBNAN

9567 The following macros shall expand to the integer constants 1 and 2, respectively;

9568 MATH_ERRNO
 9569 MATH_ERREXCEPT

9570 The following macro shall expand to an expression that has type **int** and the value
 9571 MATH_ERRNO, MATH_ERREXCEPT, or the bitwise-inclusive OR of both:

9572 math_errhandling

9573 The value of math_errhandling is constant for the duration of the program. It is unspecified
 9574 whether math_errhandling is a macro or an identifier with external linkage. If a macro definition
 9575 is suppressed or a program defines an identifier with the name math_errhandling, the behavior
 9576 is undefined. If the expression (math_errhandling & MATH_ERREXCEPT) can be non-zero, the
 9577 implementation shall define the macros FE_DIVBYZERO, FE_INVALID, and FE_OVERFLOW in
 9578 <fenv.h>.

9579 The following shall be declared as functions and may also be defined as macros. Function
 9580 prototypes shall be provided.

9581 double acos(double);
 9582 float acosf(float);
 9583 double acosh(double);
 9584 float acoshf(float);
 9585 long double acoshl(long double);
 9586 long double acosl(long double);
 9587 double asin(double);
 9588 float asinf(float);
 9589 double asinh(double);
 9590 float asinhf(float);
 9591 long double asinhl(long double);
 9592 long double asinl(long double);
 9593 double atan(double);
 9594 double atan2(double, double);
 9595 float atan2f(float, float);
 9596 long double atan2l(long double, long double);
 9597 float atanf(float);
 9598 double atanh(double);
 9599 float atanhf(float);
 9600 long double atanh1(long double);
 9601 long double atan1(long double);
 9602 double cbrt(double);
 9603 float cbrtf(float);
 9604 long double cbrt1(long double);
 9605 double ceil(double);
 9606 float ceilf(float);
 9607 long double ceill(long double);
 9608 double copysign(double, double);
 9609 float copysignf(float, float);
 9610 long double copysignl(long double, long double);
 9611 double cos(double);

Headers

<math.h>

```

9612     float      cosf(float);
9613     double     cosh(double);
9614     float      coshf(float);
9615     long double coshl(long double);
9616     long double cosl(long double);
9617     double     erf(double);
9618     double     erfc(double);
9619     float      erfcf(float);
9620     long double erfcl(long double);
9621     float      erff(float);
9622     long double erfl(long double);
9623     double     exp(double);
9624     double     exp2(double);
9625     float      exp2f(float);
9626     long double exp2l(long double);
9627     float      expf(float);
9628     long double expl(long double);
9629     double     expm1(double);
9630     float      expm1f(float);
9631     long double expm1l(long double);
9632     double     fabs(double);
9633     float      fabsf(float);
9634     long double fabsl(long double);
9635     double     fdim(double, double);
9636     float      fdimf(float, float);
9637     long double fdiml(long double, long double);
9638     double     floor(double);
9639     float      floorf(float);
9640     long double floorl(long double);
9641     double     fma(double, double, double);
9642     float      fmaf(float, float, float);
9643     long double fmal(long double, long double, long double);
9644     double     fmax(double, double);
9645     float      fmaxf(float, float);
9646     long double fmaxl(long double, long double);
9647     double     fmin(double, double);
9648     float      fminf(float, float);
9649     long double fminl(long double, long double);
9650     double     fmod(double, double);
9651     float      fmodf(float, float);
9652     long double fmodl(long double, long double);
9653     double     frexp(double, int *);
9654     float      frexpf(float, int *);
9655     long double frexpl(long double, int *);
9656     double     hypot(double, double);
9657     float      hypotf(float, float);
9658     long double hypotl(long double, long double);
9659     int        ilogb(double);
9660     int        ilogbf(float);
9661     int        ilogbl(long double);
9662     double     j0(double);
9663     double     j1(double);

```

<math.h>

Headers

```

9664 double jn(int, double);
9665 double ldexp(double, int);
9666 float ldexpf(float, int);
9667 long double ldexpl(long double, int);
9668 double lgamma(double);
9669 float lgammaf(float);
9670 long double lgammal(long double);
9671 long long llrint(double);
9672 long long llrintf(float);
9673 long long llrintl(long double);
9674 long long llround(double);
9675 long long llroundf(float);
9676 long long llroundl(long double);
9677 double log(double);
9678 double log10(double);
9679 float log10f(float);
9680 long double log10l(long double);
9681 double log1p(double);
9682 float log1pf(float);
9683 long double log1pl(long double);
9684 double log2(double);
9685 float log2f(float);
9686 long double log2l(long double);
9687 double logb(double);
9688 float logbf(float);
9689 long double logbl(long double);
9690 float logf(float);
9691 long double logl(long double);
9692 long lrint(double);
9693 long lrintf(float);
9694 long lrintl(long double);
9695 long lround(double);
9696 long lroundf(float);
9697 long lroundl(long double);
9698 double modf(double, double *);
9699 float modff(float, float *);
9700 long double modfl(long double, long double *);
9701 double nan(const char *);
9702 float nanf(const char *);
9703 long double nanl(const char *);
9704 double nearbyint(double);
9705 float nearbyintf(float);
9706 long double nearbyintl(long double);
9707 double nextafter(double, double);
9708 float nextafterf(float, float);
9709 long double nextafterl(long double, long double);
9710 double nexttoward(double, long double);
9711 float nexttowardf(float, long double);
9712 long double nexttowardl(long double, long double);
9713 double pow(double, double);
9714 float powf(float, float);
9715 long double powl(long double, long double);

```

Headers

<math.h>

```

9716     double      remainder(double, double);
9717     float        remainderf(float, float);
9718     long double  remainderl(long double, long double);
9719     double      remquo(double, double, int *);
9720     float       remquof(float, float, int *);
9721     long double remquol(long double, long double, int *);
9722     double      rint(double);
9723     float       rintf(float);
9724     long double rintl(long double);
9725     double      round(double);
9726     float       roundf(float);
9727     long double roundl(long double);
9728     double      scalbln(double, long);
9729     float       scalblnf(float, long);
9730     long double scalblnl(long double, long);
9731     double      scalbn(double, int);
9732     float       scalbnf(float, int);
9733     long double scalbnl(long double, int);
9734     double      sin(double);
9735     float       sinf(float);
9736     double      sinh(double);
9737     float       sinhlf(float);
9738     long double sinhl(long double);
9739     long double sinl(long double);
9740     double      sqrt(double);
9741     float       sqrtf(float);
9742     long double sqrtl(long double);
9743     double      tan(double);
9744     float       tanf(float);
9745     double      tanh(double);
9746     float       tanhlf(float);
9747     long double tanhl(long double);
9748     long double tanl(long double);
9749     double      tgamma(double);
9750     float       tgammaf(float);
9751     long double tgamma_l(long double);
9752     double      trunc(double);
9753     float       truncf(float);
9754     long double trunc_l(long double);
9755     double      y0(double);
9756     double      y1(double);
9757     double      yn(int, double);

```

9758 The following external variable shall be defined:

```

9759     extern int signgam;

```

9760 The behavior of each of the functions defined in <math.h> is specified in the System Interfaces
 9761 volume of POSIX.1-2008 for all representable values of its input arguments, except where stated
 9762 otherwise. Each function shall execute as if it were a single operation without generating any
 9763 externally visible exceptional conditions.

<math.h>

Headers

APPLICATION USAGE

The FP_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is off) the implementation to contract expressions. Each pragma can occur either outside external declarations or preceding all explicit declarations and statements inside a compound statement. When outside external declarations, the pragma takes effect from its occurrence until another FP_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a compound statement, the pragma takes effect from its occurrence until another FP_CONTRACT pragma is encountered (including within a nested compound statement), or until the end of the compound statement; at the end of a compound statement the state for the pragma is restored to its condition just before the compound statement. If this pragma is used in any other context, the behavior is undefined. The default state (on or off) for the pragma is implementation-defined.

Applications should use FLT_MAX as described in the <float.h> header instead of the obsolescent MAXFLOAT.

RATIONALE

Before the ISO/IEC 9899:1999 standard, the math library was defined only for the floating type **double**. All the names formed by appending 'f' or 'l' to a name in <math.h> were reserved to allow for the definition of **float** and **long double** libraries; and the ISO/IEC 9899:1999 standard provides for all three versions of math functions.

The functions *ecvt()*, *fcvt()*, and *gcvt()* have been dropped from the ISO C standard since their capability is available through *sprintf()*.

FUTURE DIRECTIONS

None.

SEE ALSO

<float.h>, <stddef.h>, <sys/types.h>

XSH Section 2.2 (on page 468), *acos()*, *acosh()*, *asin()*, *asinh()*, *atan()*, *atan2()*, *atanh()*, *cbrt()*, *ceil()*, *copysign()*, *cos()*, *cosh()*, *erf()*, *erfc()*, *exp()*, *exp2()*, *expm1()*, *fabs()*, *fdim()*, *floor()*, *fma()*, *fmax()*, *fmin()*, *fmod()*, *fpclassify()*, *frexp()*, *hypot()*, *ilogb()*, *isfinite()*, *isgreater()*, *isgreaterequal()*, *isinf()*, *isless()*, *islessequal()*, *islessgreater()*, *isnan()*, *isnormal()*, *isunordered()*, *j0()*, *ldexp()*, *lgamma()*, *llrint()*, *llround()*, *log()*, *log10()*, *log1p()*, *log2()*, *logb()*, *lrint()*, *lround()*, *modf()*, *nan()*, *nearbyint()*, *nextafter()*, *pow()*, *remainder()*, *remquo()*, *rint()*, *round()*, *scalbln()*, *signbit()*, *sin()*, *sinh()*, *sqrt()*, *tan()*, *tanh()*, *tgamma()*, *trunc()*, *y0()*

CHANGE HISTORY

First released in Issue 1.

Issue 6

This reference page is updated to align with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/21 is applied, making it clear that the meaning of the FP_FAST_FMA macro is unspecified if the macro is undefined.

Issue 7

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #47 (SD5-XBD-ERN-52) is applied, clarifying the wording of the FP_FAST_FMA macro.

The MAXFLOAT constant is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

Headers

<monetary.h>

NAME

monetary.h — monetary types

SYNOPSIS

```
#include <monetary.h>
```

DESCRIPTION

The <monetary.h> header shall define the **locale_t** type as described in <locale.h>.

The <monetary.h> header shall define the **size_t** type as described in <stddef.h>.

The <monetary.h> header shall define the **ssize_t** type as described in <sys/types.h>.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
ssize_t  strfmon(char *restrict, size_t, const char *restrict, ...);
ssize_t  strfmon_l(char *restrict, size_t, locale_t,
                  const char *restrict, ...);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<locale.h>, <stddef.h>, <sys/types.h>

XSH *strfmon()*

CHANGE HISTORY

First released in Issue 4.

Issue 6

The **restrict** keyword is added to the prototype for *strfmon()*.

Issue 7

The <monetary.h> header is moved from the XSI option to the Base.

The *strmon_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

A declaration for the **locale_t** type is added.

<mqueue.h>*Headers*9837 **NAME**9838 mqueue.h — message queues (**REALTIME**)9839 **SYNOPSIS**

9840 MSG #include <mqueue.h>

9841 **DESCRIPTION**9842 The **<mqueue.h>** header shall define the **mqd_t** type, which is used for message queue
9843 descriptors. This is not an array type.9844 The **<mqueue.h>** header shall define the **pthread_attr_t**, **size_t**, and **ssize_t** types as described
9845 in **<sys/types.h>**.9846 The **<mqueue.h>** header shall define the **struct timespec** structure as described in **<time.h>**.9847 The tag **sigevent** shall be declared as naming an incomplete structure type, the contents of which
9848 are described in the **<signal.h>** header.9849 The **<mqueue.h>** header shall define the **mq_attr** structure, which is used in getting and setting
9850 the attributes of a message queue. Attributes are initially set when the message queue is created.
9851 An **mq_attr** structure shall have at least the following fields:

9852	long	mq_flags	Message queue flags.
9853	long	mq_maxmsg	Maximum number of messages.
9854	long	mq_msgsize	Maximum message size.
9855	long	mq_curmsgs	Number of messages currently queued.

9856 The following shall be declared as functions and may also be defined as macros. Function
9857 prototypes shall be provided.

```

9858 int      mq_close(mqd_t);
9859 int      mq_getattr(mqd_t, struct mq_attr *);
9860 int      mq_notify(mqd_t, const struct sigevent *);
9861 mqd_t    mq_open(const char *, int, ...);
9862 ssize_t  mq_receive(mqd_t, char *, size_t, unsigned *);
9863 int      mq_send(mqd_t, const char *, size_t, unsigned);
9864 int      mq_setattr(mqd_t, const struct mq_attr *restrict,
9865                    struct mq_attr *restrict);
9866 ssize_t  mq_timedreceive(mqd_t, char *restrict, size_t,
9867                        unsigned *restrict, const struct timespec *restrict);
9868 int      mq_timedsend(mqd_t, const char *, size_t, unsigned,
9869                      const struct timespec *);
9870 int      mq_unlink(const char *);

```

9871 Inclusion of the **<mqueue.h>** header may make visible symbols defined in the headers
9872 **<fcntl.h>**, **<signal.h>**, and **<time.h>**.9873 **APPLICATION USAGE**

9874 None.

9875 **RATIONALE**

9876 None.

9877 **FUTURE DIRECTIONS**

9878 None.

Headers

<mqueue.h>

SEE ALSO

<fcntl.h>, <signal.h>, <sys/types.h>, <time.h>

XSH *mq_close()*, *mq_getattr()*, *mq_notify()*, *mq_open()*, *mq_receive()*, *mq_send()*, *mq_setattr()*, *mq_unlink()*

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The <mqueue.h> header is marked as part of the Message Passing option.

The *mq_timedreceive()* and *mq_timedsend()* functions are added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the prototypes for *mq_setattr()* and *mq_timedreceive()*.

Issue 7

Type and structure declarations are added.

<ndbm.h>

Headers

9892 **NAME**

9893 ndbm.h — definitions for ndbm database operations

9894 **SYNOPSIS**9895 XSI `#include <ndbm.h>`9896 **DESCRIPTION**9897 The **<ndbm.h>** header shall define the **datum** type as a structure, which shall include at least the
9898 following members:9899 `void *dptr` A pointer to the application's data.
9900 `size_t dsize` The size of the object pointed to by *dptr*.9901 The **<ndbm.h>** header shall define the **size_t** type as described in **<stddef.h>**.9902 The **<ndbm.h>** header shall define the **DBM** type.9903 The **<ndbm.h>** header shall define the following symbolic constants as possible values for the
9904 *store_mode* argument to *dbm_store()*:9905 **DBM_INSERT** Insertion of new entries only.9906 **DBM_REPLACE** Allow replacing existing entries.9907 The following shall be declared as functions and may also be defined as macros. Function
9908 prototypes shall be provided.9909 `int dbm_clearerr(DBM *);`
9910 `void dbm_close(DBM *);`
9911 `int dbm_delete(DBM *, datum);`
9912 `int dbm_error(DBM *);`
9913 `datum dbm_fetch(DBM *, datum);`
9914 `datum dbm_firstkey(DBM *);`
9915 `datum dbm_nextkey(DBM *);`
9916 `DBM *dbm_open(const char *, int, mode_t);`
9917 `int dbm_store(DBM *, datum, datum, int);`9918 The **<ndbm.h>** header shall define the **mode_t** type through **typedef**, as described in
9919 **<sys/types.h>**.9920 **APPLICATION USAGE**

9921 None.

9922 **RATIONALE**

9923 None.

9924 **FUTURE DIRECTIONS**

9925 None.

9926 **SEE ALSO**9927 **<stddef.h>**, **<sys/types.h>**9928 XSH *dbm_clearerr()*9929 **CHANGE HISTORY**

9930 First released in Issue 4, Version 2.

Headers

<ndbm.h>

- 9931 **Issue 5**
9932 References to the definitions of **size_t** and **mode_t** are added to the DESCRIPTION.
- 9933 **Issue 7**
9934 This reference page is clarified with respect to macros and symbolic constants.

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

<net/if.h>

Headers

9935 NAME

9936 net/if.h — sockets local interfaces

9937 SYNOPSIS

9938 #include <net/if.h>

9939 DESCRIPTION

9940 The <net/if.h> header shall define the **if_nameindex** structure, which shall include at least the
9941 following members:

9942 unsigned if_index Numeric index of the interface.
 9943 char *if_name Null-terminated name of the interface.

9944 The <net/if.h> header shall define the following symbolic constant for the length of a buffer
9945 containing an interface name (including the terminating NULL character):

9946 IF_NAMESIZE Interface name length.

9947 The following shall be declared as functions and may also be defined as macros. Function
9948 prototypes shall be provided.

9949 void if_freenameindex(struct if_nameindex *);
 9950 char *if_indextoname(unsigned, char *);
 9951 struct if_nameindex *if_nameindex(void);
 9952 unsigned if_nametoindex(const char *);

9953 APPLICATION USAGE

9954 None.

9955 RATIONALE

9956 None.

9957 FUTURE DIRECTIONS

9958 None.

9959 SEE ALSO

9960 XSH [if_freenameindex\(\)](#), [if_indextoname\(\)](#), [if_nameindex\(\)](#), [if_nametoindex\(\)](#)

9961 CHANGE HISTORY

9962 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9963 Issue 7

9964 This reference page is clarified with respect to macros and symbolic constants.

Headers

<netdb.h>

NAME

netdb.h — definitions for network database operations

SYNOPSIS

```
#include <netdb.h>
```

DESCRIPTION

The <netdb.h> header may define the **in_port_t** type and the **in_addr_t** type as described in <netinet/in.h>.

The <netdb.h> header shall define the **hostent** structure, which shall include at least the following members:

char	*h_name	Official name of the host.
char	**h_aliases	A pointer to an array of pointers to alternative host names, terminated by a null pointer.
int	h_addrtype	Address type.
int	h_length	The length, in bytes, of the address.
char	**h_addr_list	A pointer to an array of pointers to network addresses (in network byte order) for the host, terminated by a null pointer.

The <netdb.h> header shall define the **netent** structure, which shall include at least the following members:

char	*n_name	Official, fully-qualified (including the domain) name of the host.
char	**n_aliases	A pointer to an array of pointers to alternative network names, terminated by a null pointer.
int	n_addrtype	The address type of the network.
uint32_t	n_net	The network number, in host byte order.

The <netdb.h> header shall define the **uint32_t** type as described in <inttypes.h>.

The <netdb.h> header shall define the **protoent** structure, which shall include at least the following members:

char	*p_name	Official name of the protocol.
char	**p_aliases	A pointer to an array of pointers to alternative protocol names, terminated by a null pointer.
int	p_proto	The protocol number.

The <netdb.h> header shall define the **servent** structure, which shall include at least the following members:

char	*s_name	Official name of the service.
char	**s_aliases	A pointer to an array of pointers to alternative service names, terminated by a null pointer.
int	s_port	A value which, when converted to uint16_t , yields the port number in network byte order at which the service resides.
char	*s_proto	The name of the protocol to use when contacting the service.

<netdb.h>

Headers

10011 The <netdb.h> header shall define the IPPORT_RESERVED symbolic constant with the value of
 10012 the highest reserved Internet port number.

10013 Address Information Structure

10014 The <netdb.h> header shall define the **addrinfo** structure, which shall include at least the
 10015 following members:

10016	int	ai_flags	Input flags.
10017	int	ai_family	Address family of socket.
10018	int	ai_socktype	Socket type.
10019	int	ai_protocol	Protocol of socket.
10020	socklen_t	ai_addrlen	Length of socket address.
10021	struct sockaddr	*ai_addr	Socket address of socket.
10022	char	*ai_canonname	Canonical name of service location.
10023	struct addrinfo	*ai_next	Pointer to next in list.

10024 The <netdb.h> header shall define the following symbolic constants that evaluate to bitwise-
 10025 distinct integer constants for use in the *flags* field of the **addrinfo** structure:

10026	AI_PASSIVE	Socket address is intended for <i>bind()</i> .
10027	AI_CANONNAME	Request for canonical name.
10028	AI_NUMERICHOST	Return numeric host address as name.
10029	AI_NUMERICSERV	Inhibit service name resolution.
10030	AI_V4MAPPED	If no IPv6 addresses are found, query for IPv4 addresses and return them 10031 to the caller as IPv4-mapped IPv6 addresses.
10032	AI_ALL	Query for both IPv4 and IPv6 addresses.
10033	AI_ADDRCONFIG	Query for IPv4 addresses only when an IPv4 address is configured; query 10034 for IPv6 addresses only when an IPv6 address is configured.

10035 The <netdb.h> header shall define the following symbolic constants that evaluate to bitwise-
 10036 distinct integer constants for use in the *flags* argument to *getnameinfo()*:

10037	NI_NOFQDN	Only the nodename portion of the FQDN is returned for local hosts.
10038	NI_NUMERICHOST	The numeric form of the node's address is returned instead of its name.
10039	NI_NAMEREQD	Return an error if the node's name cannot be located in the database.
10040	NI_NUMERICSERV	The numeric form of the service address is returned instead of its name.
10041	NI_NUMERICSCOPE	
10042		For IPv6 addresses, the numeric form of the scope identifier is returned 10043 instead of its name.
10044	NI_DGRAM	Indicates that the service is a datagram service (SOCK_DGRAM).

Headers

<netdb.h>

10045 Address Information Errors

10046 The <netdb.h> header shall define the following symbolic constants for use as error values for
 10047 *getaddrinfo()* and *getnameinfo()*. The values shall be suitable for use in *#if* preprocessing
 10048 directives.

10049	EAI_AGAIN	The name could not be resolved at this time. Future attempts may
10050		succeed.
10051	EAI_BADFLAGS	The flags had an invalid value.
10052	EAI_FAIL	A non-recoverable error occurred.
10053	EAI_FAMILY	The address family was not recognized or the address length was invalid
10054		for the specified family.
10055	EAI_MEMORY	There was a memory allocation failure.
10056	EAI_NONAME	The name does not resolve for the supplied parameters.
10057		NI_NAMEREQD is set and the host's name cannot be located, or both
10058		<i>nodename</i> and <i>servname</i> were null.
10059	EAI_SERVICE	The service passed was not recognized for the specified socket type.
10060	EAI_SOCKTYPE	The intended socket type was not recognized.
10061	EAI_SYSTEM	A system error occurred. The error code can be found in <i>errno</i> .
10062	EAI_OVERFLOW	An argument buffer overflowed.

10063 The following shall be declared as functions and may also be defined as macros. Function
 10064 prototypes shall be provided.

10065	void	endhostent(void);
10066	void	endnetent(void);
10067	void	endprotoent(void);
10068	void	endservent(void);
10069	void	freeaddrinfo(struct addrinfo *);
10070	const char	*gai_strerror(int);
10071	int	getaddrinfo(const char *restrict, const char *restrict,
10072		const struct addrinfo *restrict,
10073		struct addrinfo **restrict);
10074	struct hostent	*gethostent(void);
10075	int	getnameinfo(const struct sockaddr *restrict, socklen_t,
10076		char *restrict, socklen_t, char *restrict,
10077		socklen_t, int);
10078	struct netent	*getnetbyaddr(uint32_t, int);
10079	struct netent	*getnetbyname(const char *);
10080	struct netent	*getnetent(void);
10081	struct protoent	*getprotobyname(const char *);
10082	struct protoent	*getprotobynumber(int);
10083	struct protoent	*getprotoent(void);
10084	struct servent	*getservbyname(const char *, const char *);
10085	struct servent	*getservbyport(int, const char *);
10086	struct servent	*getservent(void);
10087	void	sethostent(int);
10088	void	setnetent(int);
10089	void	setprotoent(int);

<netdb.h>

Headers

10090 void setservernt(int);

10091 The <netdb.h> header shall define the **socklen_t** type through **typedef**, as described in
10092 <sys/socket.h>.

10093 Inclusion of the <netdb.h> header may also make visible all symbols from <netinet/in.h>,
10094 <sys/socket.h>, and <inttypes.h>.

10095 APPLICATION USAGE

10096 None.

10097 RATIONALE

10098 None.

10099 FUTURE DIRECTIONS

10100 None.

10101 SEE ALSO

10102 <inttypes.h>, <netinet/in.h>, <sys/socket.h>

10103 XSH *bind()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*, *freeaddrinfo()*, *gai_strerror()*,
10104 *getnameinfo()*

10105 CHANGE HISTORY

10106 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10107 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type for
10108 *gai_strerror()* from **char *** to **const char ***. This is for coordination with the IPnG Working Group.

10109 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/11 is applied, adding a description of the
10110 NI_NUMERICSSCOPE macro and correcting the *getnameinfo()* function prototype. These changes
10111 are for alignment with IPv6.

10112 Issue 7

10113 SD5-XBD-ERN-14 is applied, changing the description of the *s_port* member of the **servent**
10114 structure.

10115 The obsolescent *h_errno* external integer, and the obsolescent *gethostbyaddr()*, and
10116 *gethostbyname()* functions are removed, along with the HOST_NOT_FOUND, NO_DATA,
10117 NO_RECOVERY, and TRY_AGAIN macros.

10118 This reference page is clarified with respect to macros and symbolic constants.

Headers

<netinet/in.h>

NAME

netinet/in.h — Internet address family

SYNOPSIS

```
#include <netinet/in.h>
```

DESCRIPTION

The <netinet/in.h> header shall define the following types:

in_port_t Equivalent to the type **uint16_t** as described in <inttypes.h>.

in_addr_t Equivalent to the type **uint32_t** as described in <inttypes.h>.

The <netinet/in.h> header shall define the **sa_family_t** type as described in <sys/socket.h>.

The <netinet/in.h> header shall define the **uint8_t** and **uint32_t** types as described in <inttypes.h>. Inclusion of the <netinet/in.h> header may also make visible all symbols from <inttypes.h> and <sys/socket.h>.

The <netinet/in.h> header shall define the **in_addr** structure, which shall include at least the following member:

```
in_addr_t    s_addr
```

The <netinet/in.h> header shall define the **sockaddr_in** structure, which shall include at least the following members:

sa_family_t	sin_family	AF_INET.
in_port_t	sin_port	Port number.
struct in_addr	sin_addr	IP address.

The *sin_port* and *sin_addr* members shall be in network byte order.

The **sockaddr_in** structure is used to store addresses for the Internet address family. Values of this type shall be cast by applications to **struct sockaddr** for use with socket functions.

IP6

The <netinet/in.h> header shall define the **in6_addr** structure, which shall include at least the following member:

```
uint8_t    s6_addr[16]
```

This array is used to contain a 128-bit IPv6 address, stored in network byte order.

The <netinet/in.h> header shall define the **sockaddr_in6** structure, which shall include at least the following members:

sa_family_t	sin6_family	AF_INET6.
in_port_t	sin6_port	Port number.
uint32_t	sin6_flowinfo	IPv6 traffic class and flow information.
struct in6_addr	sin6_addr	IPv6 address.
uint32_t	sin6_scope_id	Set of interfaces for a scope.

The *sin6_port* and *sin6_addr* members shall be in network byte order.

The **sockaddr_in6** structure shall be set to zero by an application prior to using it, since implementations are free to have additional, implementation-defined fields in **sockaddr_in6**.

The *sin6_scope_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the scope of the address carried in the *sin6_addr* field. For a link scope *sin6_addr*, the application shall ensure that *sin6_scope_id* is a link index. For a site scope *sin6_addr*, the application shall ensure that *sin6_scope_id* is a site index. The mapping of *sin6_scope_id* to an interface or set of interfaces is implementation-defined.

<netinet/in.h>

Headers

10161		The <netinet/in.h> header shall declare the following external variable:
10162		<code>const struct in6_addr in6addr_any</code>
10163		This variable is initialized by the system to contain the wildcard IPv6 address. The
10164		<netinet/in.h> header also defines the IN6ADDR_ANY_INIT macro. This macro must be
10165		constant at compile time and can be used to initialize a variable of type struct in6_addr to the
10166		IPv6 wildcard address.
10167		The <netinet/in.h> header shall declare the following external variable:
10168		<code>const struct in6_addr in6addr_loopback</code>
10169		This variable is initialized by the system to contain the loopback IPv6 address. The
10170		<netinet/in.h> header also defines the IN6ADDR_LOOPBACK_INIT macro. This macro must be
10171		constant at compile time and can be used to initialize a variable of type struct in6_addr to the
10172		IPv6 loopback address.
10173		The <netinet/in.h> header shall define the ipv6_mreq structure, which shall include at least the
10174		following members:
10175		<code>struct in6_addr</code> <code>ipv6mr_multiaddr</code> IPv6 multicast address.
10176		<code>unsigned</code> <code>ipv6mr_interface</code> Interface index.
10177		The <netinet/in.h> header shall define the following symbolic constants for use as values of the
10178		<i>level</i> argument of <i>getsockopt()</i> and <i>setsockopt()</i> :
10179		IPPROTO_IP Internet protocol.
10180	IP6	IPPROTO_IPV6 Internet Protocol Version 6.
10181		IPPROTO_ICMP Control message protocol.
10182	RS	IPPROTO_RAW Raw IP Packets Protocol.
10183		IPPROTO_TCP Transmission control protocol.
10184		IPPROTO_UDP User datagram protocol.
10185		The <netinet/in.h> header shall define the following symbolic constants for use as destination
10186		addresses for <i>connect()</i> , <i>sendmsg()</i> , and <i>sendto()</i> :
10187		INADDR_ANY IPv4 local host address.
10188		INADDR_BROADCAST IPv4 broadcast address.
10189		The <netinet/in.h> header shall define the following symbolic constant, with the value
10190		specified, to help applications declare buffers of the proper size to store IPv4 addresses in string
10191		form:
10192		INET_ADDRSTRLEN 16. Length of the string form for IP.
10193		The <i>htonl()</i> , <i>htons()</i> , <i>ntohl()</i> , and <i>ntohs()</i> functions shall be available as described in
10194		<arpa/inet.h>. Inclusion of the <netinet/in.h> header may also make visible all symbols from
10195		<arpa/inet.h>.
10196	IP6	The <netinet/in.h> header shall define the following symbolic constant, with the value
10197		specified, to help applications declare buffers of the proper size to store IPv6 addresses in string
10198		form:

Headers

<netinet/in.h>

10199	INET6_ADDRSTRLEN	46. Length of the string form for IPv6.
10200	IP6	The <netinet/in.h> header shall define the following symbolic constants, with distinct integer values, for use in the <i>option_name</i> argument in the <i>getsockopt()</i> or <i>setsockopt()</i> functions at protocol level IPPROTO_IPV6:
10201		
10202		
10203	IPV6_JOIN_GROUP	Join a multicast group.
10204	IPV6_LEAVE_GROUP	Quit a multicast group.
10205	IPV6_MULTICAST_HOPS	
10206		Multicast hop limit.
10207	IPV6_MULTICAST_IF	Interface to use for outgoing multicast packets.
10208	IPV6_MULTICAST_LOOP	
10209		Multicast packets are delivered back to the local application.
10210	IPV6_UNICAST_HOPS	Unicast hop limit.
10211	IPV6_V6ONLY	Restrict AF_INET6 socket to IPv6 communications only.
10212		The <netinet/in.h> header shall define the following macros that test for special IPv6 addresses. Each macro is of type int and takes a single argument of type const struct in6_addr * :
10213		
10214	IN6_IS_ADDR_UNSPECIFIED	
10215		Unspecified address.
10216	IN6_IS_ADDR_LOOPBACK	
10217		Loopback address.
10218	IN6_IS_ADDR_MULTICAST	
10219		Multicast address.
10220	IN6_IS_ADDR_LINKLOCAL	
10221		Unicast link-local address.
10222	IN6_IS_ADDR_SITELOCAL	
10223		Unicast site-local address.
10224	IN6_IS_ADDR_V4MAPPED	
10225		IPv4 mapped address.
10226	IN6_IS_ADDR_V4COMPAT	
10227		IPv4-compatible address.
10228	IN6_IS_ADDR_MC_NODELOCAL	
10229		Multicast node-local address.
10230	IN6_IS_ADDR_MC_LINKLOCAL	
10231		Multicast link-local address.
10232	IN6_IS_ADDR_MC_SITELOCAL	
10233		Multicast site-local address.
10234	IN6_IS_ADDR_MC_ORGLOCAL	
10235		Multicast organization-local address.
10236	IN6_IS_ADDR_MC_GLOBAL	
10237		Multicast global address.

<netinet/in.h>

Headers

10238 APPLICATION USAGE

10239 None.

10240 RATIONALE

10241 None.

10242 FUTURE DIRECTIONS

10243 None.

10244 SEE ALSO

10245 [Section 4.9](#) (on page 110), [<arpa/inet.h>](#), [<inttypes.h>](#), [<sys/socket.h>](#)10246 XSH [connect\(\)](#), [getsockopt\(\)](#), [htonl\(\)](#), [sendmsg\(\)](#), [sendto\(\)](#), [setsockopt\(\)](#)

10247 CHANGE HISTORY

10248 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10249 The *sin_zero* member was removed from the **sockaddr_in** structure as per The Open Group Base
10250 Resolution bwg2001-004.10251 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/12 is applied, adding **const** qualifiers to
10252 the *in6addr_any* and *in6addr_loopback* external variables.10253 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/22 is applied, making it clear which
10254 structure members are in network byte order.

10255 Issue 7

10256 This reference page is clarified with respect to macros and symbolic constants.

Headers

<netinet/tcp.h>

10257 **NAME**

10258 netinet/tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

10259 **SYNOPSIS**

10260 #include <netinet/tcp.h>

10261 **DESCRIPTION**

10262 The <netinet/tcp.h> header shall define the following symbolic constant for use as a socket
10263 option at the IPPROTO_TCP level:

10264 TCP_NODELAY Avoid coalescing of small segments.

10265 The implementation need not allow the value of the option to be set via *setsockopt()* or retrieved
10266 via *getsockopt()*.

10267 **APPLICATION USAGE**

10268 None.

10269 **RATIONALE**

10270 None.

10271 **FUTURE DIRECTIONS**

10272 None.

10273 **SEE ALSO**

10274 <sys/socket.h>

10275 XSH *getsockopt()*, *setsockopt()*

10276 **CHANGE HISTORY**

10277 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10278 **Issue 7**

10279 This reference page is clarified with respect to macros and symbolic constants.

<nl_types.h>

Headers

10280 **NAME**10281 `nl_types.h` — data types10282 **SYNOPSIS**10283 `#include <nl_types.h>`10284 **DESCRIPTION**10285 The **<nl_types.h>** header shall define at least the following types:

10286 **nl_catd** Used by the message catalog functions *catopen()*, *catgets()*, and *catclose()* to identify a catalog descriptor.

10288 **nl_item** Used by *nl_langinfo()* to identify items of *langinfo* data. Values of objects of type **nl_item** are defined in **<langinfo.h>**.

10290 The **<nl_types.h>** header shall define at least the following symbolic constants:

10291 **NL_SETD** Used by *genccat* when no *\$set* directive is specified in a message text source file. This constant can be passed as the value of *set_id* on subsequent calls to *catgets()* (that is, to retrieve messages from the default message set). The value of **NL_SETD** is implementation-defined.

10295 **NL_CAT_LOCALE** Value that must be passed as the *oflag* argument to *catopen()* to ensure that message catalog selection depends on the **LC_MESSAGES** locale category, rather than directly on the **LANG** environment variable.

10298 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
10300 int      catclose(nl_catd);
10301 char     *catgets(nl_catd, int, int, const char *);
10302 nl_catd  catopen(const char *, int);
```

10303 **APPLICATION USAGE**

10304 None.

10305 **RATIONALE**

10306 None.

10307 **FUTURE DIRECTIONS**

10308 None.

10309 **SEE ALSO**10310 [**<langinfo.h>**](#)10311 XSH *catclose()*, *catgets()*, *catopen()*, *nl_langinfo()*10312 XCU *genccat*10313 **CHANGE HISTORY**

10314 First released in Issue 2.

10315 **Issue 7**10316 The **<nl_types.h>** header is moved from the XSI option to the Base.

10317 This reference page is clarified with respect to macros and symbolic constants.

Headers

<poll.h>

NAME

poll.h — definitions for the poll() function

SYNOPSIS

```
#include <poll.h>
```

DESCRIPTION

The <poll.h> header shall define the **pollfd** structure, which shall include at least the following members:

int	fd	The following descriptor being polled.
short	events	The input event flags (see below).
short	revents	The output event flags (see below).

The <poll.h> header shall define the following type through **typedef**:

nfds_t An unsigned integer type used for the number of file descriptors.

The implementation shall support one or more programming environments in which the width of **nfds_t** is no greater than the width of type **long**. The names of these programming environments can be obtained using the *confstr()* function or the *getconf* utility.

The <poll.h> header shall define the following symbolic constants, zero or more of which may be OR'ed together to form the *events* or *revents* members in the **pollfd** structure:

POLLIN	Data other than high-priority data may be read without blocking.
POLLRDNORM	Normal data may be read without blocking.
POLLRDBAND	Priority data may be read without blocking.
POLLPRI	High priority data may be read without blocking.
POLLOUT	Normal data may be written without blocking.
POLLWRNORM	Equivalent to POLLOUT.
POLLWRBAND	Priority data may be written.
POLLERR	An error has occurred (<i>revents</i> only).
POLLHUP	Device has been disconnected (<i>revents</i> only).
POLLNVAL	Invalid <i>fd</i> member (<i>revents</i> only).

The significance and semantics of normal, priority, and high-priority data are file and device-specific.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
int poll(struct pollfd [], nfds_t, int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

<poll.h>*Headers*10356 **SEE ALSO**10357 XSH *confstr()*, *poll()*10358 XCU *getconf*10359 **CHANGE HISTORY**

10360 First released in Issue 4, Version 2.

10361 **Issue 6**10362 The description of the symbolic constants is updated to match the *poll()* function.10363 Text related to STREAMS has been moved to the *poll()* reference page.10364 A note is added to the DESCRIPTION regarding the significance and semantics of normal,
10365 priority, and high-priority data.10366 **Issue 7**10367 The **<poll.h>** header is moved from the XSI option to the Base.

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

Headers

<pthread.h>

NAME

pthread.h — threads

SYNOPSIS

#include <pthread.h>

DESCRIPTION

The <pthread.h> header shall define the following symbolic constants:

PTHREAD_BARRIER_SERIAL_THREAD
 PTHREAD_CANCEL_ASYNCHRONOUS
 PTHREAD_CANCEL_ENABLE
 PTHREAD_CANCEL_DEFERRED
 PTHREAD_CANCEL_DISABLE
 PTHREAD_CANCELED
 PTHREAD_CREATE_DETACHED
 PTHREAD_CREATE_JOINABLE
 PTHREAD_EXPLICIT_SCHED
 PTHREAD_INHERIT_SCHED
 PTHREAD_MUTEX_DEFAULT
 PTHREAD_MUTEX_ERRORCHECK
 PTHREAD_MUTEX_NORMAL
 PTHREAD_MUTEX_RECURSIVE
 PTHREAD_MUTEX_ROBUST
 PTHREAD_MUTEX_STALLED
 PTHREAD_ONCE_INIT
 PTHREAD_PRIO_INHERIT
 PTHREAD_PRIO_NONE
 PTHREAD_PRIO_PROTECT
 PTHREAD_PROCESS_SHARED
 PTHREAD_PROCESS_PRIVATE
 PTHREAD_SCOPE_PROCESS
 PTHREAD_SCOPE_SYSTEM

The <pthread.h> header shall define the following compile-time constant expressions valid as initializers for the following types:

Name	Initializer for Type
PTHREAD_COND_INITIALIZER	pthread_cond_t
PTHREAD_MUTEX_INITIALIZER	pthread_mutex_t
PTHREAD_RWLOCK_INITIALIZER	pthread_rwlock_t

The <pthread.h> header shall define the pthread_attr_t, pthread_barrier_t, pthread_barrierattr_t, pthread_cond_t, pthread_condattr_t, pthread_key_t, pthread_mutex_t, pthread_mutexattr_t, pthread_once_t, pthread_rwlock_t, pthread_rwlockattr_t, pthread_spinlock_t, and pthread_t types as described in <sys/types.h>.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```

int pthread_atfork(void (*)(void), void (*)(void),
void (*)(void));
int pthread_attr_destroy(pthread_attr_t *);
int pthread_attr_getdetachstate(const pthread_attr_t *, int *);
int pthread_attr_getguardsize(const pthread_attr_t *restrict,
```

<pthread.h>

Headers

```

10415         size_t *restrict);
10416 TPS     int  pthread_attr_getinheritsched(const pthread_attr_t *restrict,
10417         int *restrict);
10418         int  pthread_attr_getschedparam(const pthread_attr_t *restrict,
10419         struct sched_param *restrict);
10420 TPS     int  pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
10421         int *restrict);
10422         int  pthread_attr_getscope(const pthread_attr_t *restrict,
10423         int *restrict);
10424 TSA TSS int  pthread_attr_getstack(const pthread_attr_t *restrict,
10425         void **restrict, size_t *restrict);
10426 TSS     int  pthread_attr_getstacksize(const pthread_attr_t *restrict,
10427         size_t *restrict);
10428         int  pthread_attr_init(pthread_attr_t *);
10429         int  pthread_attr_setdetachstate(pthread_attr_t *, int);
10430         int  pthread_attr_setguardsize(pthread_attr_t *, size_t);
10431 TPS     int  pthread_attr_setinheritsched(pthread_attr_t *, int);
10432         int  pthread_attr_setschedparam(pthread_attr_t *restrict,
10433         const struct sched_param *restrict);
10434 TPS     int  pthread_attr_setschedpolicy(pthread_attr_t *, int);
10435         int  pthread_attr_setscope(pthread_attr_t *, int);
10436 TSA TSS int  pthread_attr_setstack(pthread_attr_t *, void *, size_t);
10437 TSS     int  pthread_attr_setstacksize(pthread_attr_t *, size_t);
10438         int  pthread_barrier_destroy(pthread_barrier_t *);
10439         int  pthread_barrier_init(pthread_barrier_t *restrict,
10440         const pthread_barrierattr_t *restrict, unsigned);
10441         int  pthread_barrier_wait(pthread_barrier_t *);
10442         int  pthread_barrierattr_destroy(pthread_barrierattr_t *);
10443 TSH     int  pthread_barrierattr_getpshared(
10444         const pthread_barrierattr_t *restrict, int *restrict);
10445         int  pthread_barrierattr_init(pthread_barrierattr_t *);
10446 TSH     int  pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
10447         int  pthread_cancel(pthread_t);
10448         void pthread_cleanup_pop(int);
10449         void pthread_cleanup_push(void (*)(void*), void *);
10450         int  pthread_cond_broadcast(pthread_cond_t *);
10451         int  pthread_cond_destroy(pthread_cond_t *);
10452         int  pthread_cond_init(pthread_cond_t *restrict,
10453         const pthread_condattr_t *restrict);
10454         int  pthread_cond_signal(pthread_cond_t *);
10455         int  pthread_cond_timedwait(pthread_cond_t *restrict,
10456         pthread_mutex_t *restrict, const struct timespec *restrict);
10457         int  pthread_cond_wait(pthread_cond_t *restrict,
10458         pthread_mutex_t *restrict);
10459         int  pthread_condattr_destroy(pthread_condattr_t *);
10460         int  pthread_condattr_getclock(const pthread_condattr_t *restrict,
10461         clockid_t *restrict);
10462 TSH     int  pthread_condattr_getpshared(const pthread_condattr_t *restrict,
10463         int *restrict);
10464         int  pthread_condattr_init(pthread_condattr_t *);
10465         int  pthread_condattr_setclock(pthread_condattr_t *, clockid_t);

```

Headers

<pthread.h>

```

10466 TSH      int      pthread_condattr_setpshared(pthread_condattr_t *, int);
10467          int      pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
10468                                void *(*)(void*), void *restrict);
10469          int      pthread_detach(pthread_t);
10470          int      pthread_equal(pthread_t, pthread_t);
10471          void     pthread_exit(void *);
10472 OB XSI     int      pthread_getconcurrency(void);
10473 TCT        int      pthread_getcpuclockid(pthread_t, clockid_t *);
10474 TPS        int      pthread_getschedparam(pthread_t, int *restrict,
10475                                struct sched_param *restrict);
10476          void     *pthread_getspecific(pthread_key_t);
10477          int      pthread_join(pthread_t, void **);
10478          int      pthread_key_create(pthread_key_t *, void (*)(void*));
10479          int      pthread_key_delete(pthread_key_t);
10480          int      pthread_mutex_consistent(pthread_mutex_t *);
10481          int      pthread_mutex_destroy(pthread_mutex_t *);
10482 RPP|TPP    int      pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
10483                                int *restrict);
10484          int      pthread_mutex_init(pthread_mutex_t *restrict,
10485                                const pthread_mutexattr_t *restrict);
10486          int      pthread_mutex_lock(pthread_mutex_t *);
10487 RPP|TPP    int      pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
10488                                int *restrict);
10489          int      pthread_mutex_timedlock(pthread_mutex_t *restrict,
10490                                const struct timespec *restrict);
10491          int      pthread_mutex_trylock(pthread_mutex_t *);
10492          int      pthread_mutex_unlock(pthread_mutex_t *);
10493          int      pthread_mutexattr_destroy(pthread_mutexattr_t *);
10494 RPP|TPP    int      pthread_mutexattr_getprioceiling(
10495                                const pthread_mutexattr_t *restrict, int *restrict);
10496 MC1        int      pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict,
10497                                int *restrict);
10498 TSH        int      pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
10499                                int *restrict);
10500          int      pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict,
10501                                int *restrict);
10502          int      pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
10503                                int *restrict);
10504          int      pthread_mutexattr_init(pthread_mutexattr_t *);
10505 RPP|TPP    int      pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
10506 MC1        int      pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
10507 TSH        int      pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
10508          int      pthread_mutexattr_setrobust(pthread_mutexattr_t *, int);
10509          int      pthread_mutexattr_settype(pthread_mutexattr_t *, int);
10510          int      pthread_once(pthread_once_t *, void (*)(void));
10511          int      pthread_rwlock_destroy(pthread_rwlock_t *);
10512          int      pthread_rwlock_init(pthread_rwlock_t *restrict,
10513                                const pthread_rwlockattr_t *restrict);
10514          int      pthread_rwlock_rdlock(pthread_rwlock_t *);
10515          int      pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
10516                                const struct timespec *restrict);
10517          int      pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,

```

<pthread.h>

Headers

```

10518         const struct timespec *restrict);
10519 int     pthread_rwlock_tryrdlock(pthread_rwlock_t *);
10520 int     pthread_rwlock_trywrlock(pthread_rwlock_t *);
10521 int     pthread_rwlock_unlock(pthread_rwlock_t *);
10522 int     pthread_rwlock_wrlock(pthread_rwlock_t *);
10523 int     pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
10524 TSH    int     pthread_rwlockattr_getpshared(
10525         const pthread_rwlockattr_t *restrict, int *restrict);
10526 int     pthread_rwlockattr_init(pthread_rwlockattr_t *);
10527 TSH    int     pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
10528 pthread_t
10529         pthread_self(void);
10530 int     pthread_setcancelstate(int, int *);
10531 int     pthread_setcanceltype(int, int *);
10532 OB XSI int     pthread_setconcurrency(int);
10533 TPS    int     pthread_setschedparam(pthread_t, int,
10534         const struct sched_param *);
10535 int     pthread_setschedprio(pthread_t, int);
10536 int     pthread_setspecific(pthread_key_t, const void *);
10537 int     pthread_spin_destroy(pthread_spinlock_t *);
10538 int     pthread_spin_init(pthread_spinlock_t *, int);
10539 int     pthread_spin_lock(pthread_spinlock_t *);
10540 int     pthread_spin_trylock(pthread_spinlock_t *);
10541 int     pthread_spin_unlock(pthread_spinlock_t *);
10542 void    pthread_testcancel(void);

```

10543 Inclusion of the <pthread.h> header shall make symbols defined in the headers <sched.h> and
10544 <time.h> visible.

10545 APPLICATION USAGE

10546 None.

10547 RATIONALE

10548 None.

10549 FUTURE DIRECTIONS

10550 None.

10551 SEE ALSO

10552 <sched.h>, <sys/types.h>, <time.h>

10553 XSH pthread_atfork(), pthread_attr_destroy(), pthread_attr_getdetachstate(),
10554 pthread_attr_getguardsize(), pthread_attr_getinheritsched(), pthread_attr_getschedparam(),
10555 pthread_attr_getschedpolicy(), pthread_attr_getscope(), pthread_attr_getstack(),
10556 pthread_attr_getstacksize(), pthread_barrier_destroy(), pthread_barrier_wait(),
10557 pthread_barrierattr_destroy(), pthread_barrierattr_getpshared(), pthread_cancel(),
10558 pthread_cleanup_pop(), pthread_cond_broadcast(), pthread_cond_destroy(), pthread_cond_timedwait(),
10559 pthread_condattr_destroy(), pthread_condattr_getclock(), pthread_condattr_getpshared(),
10560 pthread_create(), pthread_detach(), pthread_equal(), pthread_exit(), pthread_getconcurrency(),
10561 pthread_getcpuclockid(), pthread_getschedparam(), pthread_getspecific(), pthread_join(),
10562 pthread_key_create(), pthread_key_delete(), pthread_mutex_consistent(), pthread_mutex_destroy(),
10563 pthread_mutex_getprioceiling(), pthread_mutex_lock(), pthread_mutex_timedlock(),
10564 pthread_mutexattr_destroy(), pthread_mutexattr_getprioceiling(), pthread_mutexattr_getprotocol(),
10565 pthread_mutexattr_getpshared(), pthread_mutexattr_getrobust(), pthread_mutexattr_gettype(),
10566 pthread_once(), pthread_rwlock_destroy(), pthread_rwlock_rdlock(), pthread_rwlock_timedrdlock(),

Headers

<pthread.h>

*pthread_rwlock_timedwrlock(), pthread_rwlock_trywrlock(), pthread_rwlock_unlock(),
pthread_rwlockattr_destroy(), pthread_rwlockattr_getpshared(), pthread_self(),
pthread_setcancelstate(), pthread_setschedprio(), pthread_spin_destroy(), pthread_spin_lock(),
pthread_spin_unlock()*

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The RTT margin markers are broken out into their POSIX options.

The Open Group Corrigendum U021/9 is applied, correcting the prototype for the *pthread_cond_wait()* function.

The Open Group Corrigendum U026/2 is applied, correcting the prototype for the *pthread_setschedparam()* function so that its second argument is of type **int**.

The *pthread_getcpuclockid()* and *pthread_mutex_timedlock()* functions are added for alignment with IEEE Std 1003.1d-1999.

The following functions are added for alignment with IEEE Std 1003.1j-2000:

*pthread_barrier_destroy(), pthread_barrier_init(), pthread_barrier_wait(),
pthread_barrierattr_destroy(), pthread_barrierattr_getpshared(), pthread_barrierattr_init(),
pthread_barrierattr_setpshared(), pthread_condattr_getclock(), pthread_condattr_setclock(),
pthread_rwlock_timedrdlock(), pthread_rwlock_timedwrlock(), pthread_spin_destroy(),
pthread_spin_init(), pthread_spin_lock(), pthread_spin_trylock(), and pthread_spin_unlock().*

PTHREAD_RWLOCK_INITIALIZER is removed for alignment with IEEE Std 1003.1j-2000.

Functions previously marked as part of the Read-Write Locks option are now moved to the Threads option.

The **restrict** keyword is added to the prototypes for *pthread_attr_getguardsize(),
pthread_attr_getinheritsched(), pthread_attr_getschedparam(), pthread_attr_getschedpolicy(),
pthread_attr_getscope(), pthread_attr_getstackaddr(), pthread_attr_getstacksize(),
pthread_attr_setschedparam(), pthread_barrier_init(), pthread_barrierattr_getpshared(),
pthread_cond_init(), pthread_cond_signal(), pthread_cond_timedwait(), pthread_cond_wait(),
pthread_condattr_getclock(), pthread_condattr_getpshared(), pthread_create(),
pthread_getschedparam(), pthread_mutex_getprioceiling(), pthread_mutex_init(),
pthread_mutex_setprioceiling(), pthread_mutexattr_getprioceiling(), pthread_mutexattr_getprotocol(),
pthread_mutexattr_getpshared(), pthread_mutexattr_gettype(), pthread_rwlock_init(),
pthread_rwlock_timedrdlock(), pthread_rwlock_timedwrlock(), pthread_rwlockattr_getpshared(), and
pthread_sigmask().*

IEEE PASC Interpretation 1003.1 #86 is applied, allowing the symbols from <sched.h> and <time.h> to be made visible when <pthread.h> is included. Previously this was an XSI option.

IEEE PASC Interpretation 1003.1c #42 is applied, removing the requirement for prototypes for the *pthread_kill()* and *pthread_sigmask()* functions. These are required to be in the <signal.h> header. They are allowed here through the name space rules.

IEEE PASC Interpretation 1003.1 #96 is applied, adding the *pthread_setschedprio()* function.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/13 is applied, correcting shading errors that were in contradiction with the System Interfaces volume of POSIX.1-2008.

<pthread.h>

Headers

10609 **Issue 7**

10610 SD5-XBD-ERN-55 is applied, adding the **restrict** keyword to the *pthread_mutex_timedlock()*
 10611 function prototype.

10612 SD5-XBD-ERN-62 is applied.

10613 Austin Group Interpretation 1003.1-2001 #048 is applied, reinstating the
 10614 PTHREAD_RWLOCK_INITIALIZER symbol.

10615 The <pthread.h> header is moved from the Threads option to the Base.

10616 The following extended mutex types are moved from the XSI option to the Base:

10617 PTHREAD_MUTEX_NORMAL
 10618 PTHREAD_MUTEX_ERRORCHECK
 10619 PTHREAD_MUTEX_RECURSIVE
 10620 PTHREAD_MUTEX_DEFAULT

10621 The PTHREAD_MUTEX_ROBUST and PTHREAD_MUTEX_STALLED symbols and the
 10622 *pthread_mutex_consistent()*, *pthread_mutexattr_getrobust()*, and *pthread_mutexattr_setrobust()*
 10623 functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

10624 Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options
 10625 is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex
 10626 or Robust Mutex Priority Inheritance, respectively.

10627 This reference page is clarified with respect to macros and symbolic constants.

Headers

<pwd.h>

NAME

pwd.h — password structure

SYNOPSIS

```
#include <pwd.h>
```

DESCRIPTION

The <pwd.h> header shall define the **struct passwd**, structure, which shall include at least the following members:

char	*pw_name	User's login name.
uid_t	pw_uid	Numerical user ID.
gid_t	pw_gid	Numerical group ID.
char	*pw_dir	Initial working directory.
char	*pw_shell	Program to use as shell.

The <pwd.h> header shall define the **gid_t**, **uid_t**, and **size_t** types as described in <sys/types.h>.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void      endpwent(void);
struct passwd *getpwent(void);
struct passwd *getpwnam(const char *);
int      getpwnam_r(const char *, struct passwd *, char *,
                  size_t, struct passwd **);
struct passwd *getpwuid(uid_t);
int      getpwuid_r(uid_t, struct passwd *, char *,
                  size_t, struct passwd **);
void      setpwent(void);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *endpwent()*, *getpwnam()*, *getpwuid()*

CHANGE HISTORY

First released in Issue 1.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **gid_t** and **uid_t** types are mandated.

<pwd.h>

Headers

- 10670 • The *getpwnam_r()* and *getpwuid_r()* functions are marked as part of the Thread-Safe
10671 Functions option.

10672 **Issue 7**

10673 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size_t** type.

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

Headers

<regex.h>

NAME

regex.h — regular expression matching types

SYNOPSIS

```
#include <regex.h>
```

DESCRIPTION

The <regex.h> header shall define the structures and symbolic constants used by the *regcomp()*, *regex()*, *regerror()*, and *regfree()* functions.

The <regex.h> header shall define the **regex_t** structure type, which shall include at least the following member:

size_t	re_nsub	Number of parenthesized subexpressions.

The <regex.h> header shall define the **size_t** type as described in <sys/types.h>.

The <regex.h> header shall define the **regoff_t** type as a signed integer type that can hold the largest value that can be stored in either a **ptrdiff_t** type or a **ssize_t** type.

The <regex.h> header shall define the **regmatch_t** structure type, which shall include at least the following members:

regoff_t	rm_so	Byte offset from start of string to start of substring.
regoff_t	rm_eo	Byte offset from start of string of the first character after the end of substring.

The <regex.h> header shall define the following symbolic constants for the *cflags* parameter to the *regcomp()* function:

REG_EXTENDED	Use Extended Regular Expressions.
REG_ICASE	Ignore case in match.
REG_NOSUB	Report only success or fail in <i>regex()</i> .
REG_NEWLINE	Change the handling of <newline>.

The <regex.h> header shall define the following symbolic constants for the *eflags* parameter to the *regex()* function:

REG_NOTBOL	The <circumflex> character ('^'), when taken as a special character, does not match the beginning of <i>string</i> .
REG_NOTEOL	The <dollar-sign> ('\$'), when taken as a special character, does not match the end of <i>string</i> .

The <regex.h> header shall define the following symbolic constants as error return values:

REG_NOMATCH	<i>regex()</i> failed to match.
REG_BADPAT	Invalid regular expression.
REG_ECOLLATE	Invalid collating element referenced.
REG_ETYPE	Invalid character class type referenced.
REG_ESCAPE	Trailing <backslash> character in pattern.
REG_ESUBREG	Number in \digit invalid or in error.

<regex.h>

Headers

10712	REG_EBRACK	"[]" imbalance.
10713	REG_EPAREN	"\(\)" or "()" imbalance.
10714	REG_EBRACE	"\{\}" imbalance.
10715	REG_BADBR	Content of "\{\}" invalid: not a number, number too large, more than
10716		two numbers, first larger than second.
10717	REG_ERANGE	Invalid endpoint in range expression.
10718	REG_ESPACE	Out of memory.
10719	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.
10720	The following shall be declared as functions and may also be defined as macros. Function	
10721	prototypes shall be provided.	
10722	<code>int regcomp(regex_t *restrict, const char *restrict, int);</code>	
10723	<code>size_t regerror(int, const regex_t *restrict, char *restrict, size_t);</code>	
10724	<code>int regexec(const regex_t *restrict, const char *restrict, size_t,</code>	
10725	<code>regmatch_t [restrict], int);</code>	
10726	<code>void regfree(regex_t *);</code>	
10727	The implementation may define additional macros or constants using names beginning with	
10728	REG_.	
10729	APPLICATION USAGE	
10730	None.	
10731	RATIONALE	
10732	None.	
10733	FUTURE DIRECTIONS	
10734	None.	
10735	SEE ALSO	
10736	<sys/types.h>	
10737	XSH <i>regcomp()</i>	
10738	CHANGE HISTORY	
10739	First released in Issue 4.	
10740	Originally derived from the ISO POSIX-2 standard.	
10741	Issue 6	
10742	The REG_ENOSYS constant is marked obsolescent.	
10743	The <i>restrict</i> keyword is added to the prototypes for <i>regcomp()</i> , <i>regerror()</i> , and <i>regexec()</i> .	
10744	A statement is added that the <i>size_t</i> type is defined as described in <sys/types.h>.	
10745	Issue 7	
10746	SD5-XBD-ERN-60 is applied.	
10747	The obsolescent REG_ENOSYS constant is removed.	
10748	This reference page is clarified with respect to macros and symbolic constants.	

Headers

<sched.h>

10749 NAME

10750 sched.h — execution scheduling

10751 SYNOPSIS

10752 #include <sched.h>

10753 DESCRIPTION

10754 PS The <sched.h> header shall define the **pid_t** type as described in <sys/types.h>.10755 SS|TSP The <sched.h> header shall define the **time_t** type as described in <sys/types.h>.10756 The <sched.h> header shall define the **timespec** structure as described in <time.h>.

10757 The <sched.h> header shall define the **sched_param** structure, which shall include the
 10758 scheduling parameters required for implementation of each supported scheduling policy. This
 10759 structure shall include at least the following member:

10760 int sched_priority Process or thread execution scheduling priority.

10761 SS|TSP The **sched_param** structure defined in <sched.h> shall include the following members in
 10762 addition to those specified above:

10763	int	sched_ss_low_priority	Low scheduling priority for
10764			sporadic server.
10765	struct timespec	sched_ss_repl_period	Replenishment period for
10766			sporadic server.
10767	struct timespec	sched_ss_init_budget	Initial budget for sporadic server.
10768	int	sched_ss_max_repl	Maximum pending replenishments for
10769			sporadic server.

10770 Each process or thread is controlled by an associated scheduling policy and priority. Associated
 10771 with each policy is a priority range. Each policy definition specifies the minimum priority range
 10772 for that policy. The priority ranges for each policy may overlap the priority ranges of other
 10773 policies.

10774 Four scheduling policies are defined; others may be defined by the implementation. The four
 10775 standard policies are indicated by the values of the following symbolic constants:

10776 PS|TPS **SCHED_FIFO** First in-first out (FIFO) scheduling policy.

10777 PS|TPS **SCHED_RR** Round robin scheduling policy.

10778 SS|TSP **SCHED_SPORADIC** Sporadic server scheduling policy.

10779 PS|TPS **SCHED_OTHER** Another scheduling policy.

10780 The values of these constants are distinct.

10781 The following shall be declared as functions and may also be defined as macros. Function
 10782 prototypes shall be provided.

10783	PS TPS	int	sched_get_priority_max(int);
10784		int	sched_get_priority_min(int);
10785	PS	int	sched_getparam(pid_t, struct sched_param *);
10786		int	sched_getscheduler(pid_t);
10787	PS TPS	int	sched_rr_get_interval(pid_t, struct timespec *);
10788	PS	int	sched_setparam(pid_t, const struct sched_param *);
10789		int	sched_setscheduler(pid_t, int, const struct sched_param *);
10790		int	sched_yield(void);

<sched.h>

Headers

10791 Inclusion of the <sched.h> header may make visible all symbols from the <time.h> header.

10792 **APPLICATION USAGE**

10793 None.

10794 **RATIONALE**

10795 None.

10796 **FUTURE DIRECTIONS**

10797 None.

10798 **SEE ALSO**

10799 <sys/types.h>, <time.h>

10800 XSH *sched_get_priority_max()*, *sched_getparam()*, *sched_getscheduler()*, *sched_rr_get_interval()*,
10801 *sched_setparam()*, *sched_setscheduler()*, *sched_yield()*

10802 **CHANGE HISTORY**

10803 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10804 **Issue 6**

10805 The <sched.h> header is marked as part of the Process Scheduling option.

10806 Sporadic server members are added to the **sched_param** structure, and the SCHED_SPORADIC
10807 scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

10808 IEEE PASC Interpretation 1003.1 #108 is applied, correcting the **sched_param** structure whose
10809 members *sched_ss_repl_period* and *sched_ss_init_budget* should be type **struct timespec** and not
10810 **timespec**.

10811 Symbols from <time.h> may be made visible when <sched.h> is included.

10812 IEEE Std 1003.1-2001/Cor 1-2002, items XSH/TC1/D6/52 and XSH/TC1/D6/53 are applied,
10813 aligning the function prototype shading and margin codes with the System Interfaces volume of
10814 IEEE Std 1003.1-2001.

10815 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/23 is applied, updating the
10816 DESCRIPTION to differentiate between thread and process execution.

10817 **Issue 7**

10818 SD5-XBD-ERN-13 is applied.

10819 Austin Group Interpretation 1003.1-2001 #064 is applied, correcting the options markings.

10820 The <sched.h> headers is moved from the Threads option to the Base.

10821 Declarations for the **pid_t** and **time_t** types and the **timespec** structure are added.

Headers

<search.h>

NAME

search.h — search tables

SYNOPSIS

XSI `#include <search.h>`

DESCRIPTION

The **<search.h>** header shall define the **ENTRY** type for structure **entry** which shall include the following members:

```
char    *key
void    *data
```

and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as follows:

```
enum { FIND, ENTER } ACTION;
enum { preorder, postorder, endorder, leaf } VISIT;
```

The **<search.h>** header shall define the **size_t** type as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int      hcreate(size_t);
void     hdestroy(void);
ENTRY *hsearch(ENTRY, ACTION);
void     insque(void *, void *);
void     *lfind(const void *, const void *, size_t *,
               size_t, int (*)(const void *, const void *));
void     *lsearch(const void *, void *, size_t *,
               size_t, int (*)(const void *, const void *));
void     remque(void *);
void     *tdelete(const void *restrict, void **restrict,
               int (*)(const void *, const void *));
void     *tfind(const void *, void *const *,
               int (*)(const void *, const void *));
void     *tsearch(const void *, void **,
               int (*)(const void *, const void *));
void     twalk(const void *,
               void (*)(const void *, VISIT, int ));
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[**<sys/types.h>**](#)XSH [*hcreate\(\)*](#), [*insque\(\)*](#), [*lsearch\(\)*](#), [*tdelete\(\)*](#)

<search.h>

Headers

10864 **CHANGE HISTORY**

10865 First released in Issue 1. Derived from Issue 1 of the SVID.

10866 **Issue 6**10867 The Open Group Corrigendum U021/6 is applied, updating the prototypes for *tdelete()* and
10868 *tsearch()*.10869 The **restrict** keyword is added to the prototype for *tdelete()*.

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

Headers

<semaphore.h>

NAME

semaphore.h — semaphores

SYNOPSIS

```
#include <semaphore.h>
```

DESCRIPTION

The **<semaphore.h>** header shall define the **sem_t** type, used in performing semaphore operations. The semaphore may be implemented using a file descriptor, in which case applications are able to open up at least a total of {OPEN_MAX} files and semaphores.

The **<semaphore.h>** header shall define the symbolic constant SEM_FAILED which shall have type **sem_t ***.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int      sem_close(sem_t *);
int      sem_destroy(sem_t *);
int      sem_getvalue(sem_t *restrict, int *restrict);
int      sem_init(sem_t *, int, unsigned);
sem_t *  sem_open(const char *, int, ...);
int      sem_post(sem_t *);
int      sem_timedwait(sem_t *restrict, const struct timespec *restrict);
int      sem_trywait(sem_t *);
int      sem_unlink(const char *);
int      sem_wait(sem_t *);
```

Inclusion of the **<semaphore.h>** header may make visible symbols defined in the **<fcntl.h>** and **<time.h>** headers.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<fcntl.h>, **<sys/types.h>**, **<time.h>**

XSH *sem_close()*, *sem_destroy()*, *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_timedwait()*, *sem_trywait()*, *sem_unlink()*

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The **<semaphore.h>** header is marked as part of the Semaphores option.

The Open Group Corrigendum U021/3 is applied, adding a description of SEM_FAILED.

The *sem_timedwait()* function is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the prototypes for *sem_getvalue()* and *sem_timedwait()*.

<semaphore.h>

Headers

10911 **Issue 7**

10912 SD5-XBD-ERN-57 is applied, allowing the header to make visible symbols from the <time.h>
10913 header.

10914 The <semaphore.h> header is moved from the Semaphores option to the Base.

10915 This reference page is clarified with respect to macros and symbolic constants.

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

Headers

<setjmp.h>

10916 **NAME**

10917 setjmp.h — stack environment declarations

10918 **SYNOPSIS**

10919 #include <setjmp.h>

10920 **DESCRIPTION**

10921 CX Some of the functionality described on this reference page extends the ISO C standard.
 10922 Applications shall define the appropriate feature test macro (see XSH [Section 2.2](#), on page 468) to
 10923 enable the visibility of these symbols in this header.

10924 CX The <setjmp.h> header shall define the array types **jmp_buf** and **sigjmp_buf**.

10925 The following shall be declared as functions and may also be defined as macros. Function
 10926 prototypes shall be provided.

10927 OB XSI void _longjmp(jmp_buf, int);

10928 void longjmp(jmp_buf, int);

10929 CX void siglongjmp(sigjmp_buf, int);

10930 The following may be declared as functions, or defined as macros, or both. If functions are
 10931 declared, function prototypes shall be provided.

10932 OB XSI int _setjmp(jmp_buf);

10933 int setjmp(jmp_buf);

10934 CX int sigsetjmp(sigjmp_buf, int);

10935 **APPLICATION USAGE**

10936 None.

10937 **RATIONALE**

10938 None.

10939 **FUTURE DIRECTIONS**

10940 None.

10941 **SEE ALSO**

10942 XSH [Section 2.2](#) (on page 468), [_longjmp\(\)](#), [longjmp\(\)](#), [setjmp\(\)](#), [siglongjmp\(\)](#), [sigsetjmp\(\)](#)

10943 **CHANGE HISTORY**

10944 First released in Issue 1.

10945 **Issue 6**

10946 Extensions beyond the ISO C standard are marked.

10947 **Issue 7**

10948 SD5-XBD-ERN-6 is applied.

<signal.h>

Headers

10949 NAME

10950 signal.h — signals

10951 SYNOPSIS

10952 #include <signal.h>

10953 DESCRIPTION

10954 CX Some of the functionality described on this reference page extends the ISO C standard.
 10955 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to
 10956 enable the visibility of these symbols in this header.

10957 The <signal.h> header shall define the following macros, which shall expand to constant
 10958 expressions with distinct values that have a type compatible with the second argument to, and
 10959 the return value of, the *signal()* function, and whose values shall compare unequal to the
 10960 address of any declarable function.

10961 SIG_DFL Request for default signal handling.

10962 SIG_ERR Return value from *signal()* in case of error.

10963 OB CX SIG_HOLD Request that signal be held.

10964 SIG_IGN Request that signal be ignored.

10965 CX The <signal.h> header shall define the **pthread_t**, **size_t**, and **uid_t** types as described in
 10966 <sys/types.h>.

10967 The <signal.h> header shall define the **timespec** structure as described in <time.h>.

10968 The <signal.h> header shall define the following data types:

10969 **sig_atomic_t** Possibly volatile-qualified integer type of an object that can be accessed as
 10970 an atomic entity, even in the presence of asynchronous interrupts.

10971 CX **sigset_t** Integer or structure type of an object used to represent sets of signals.

10972 CX **pid_t** As described in <sys/types.h>.

10973 CX The <signal.h> header shall define the **pthread_attr_t** type as described in <sys/types.h>.

10974 The <signal.h> header shall define the **sigevent** structure, which shall include at least the
 10975 following members:

10976	int	sigev_notify	Notification type.
10977	int	sigev_signo	Signal number.
10978	union sigval	sigev_value	Signal value.
10979	void	(*sigev_notify_function) (union sigval)	Notification function.
10980			Notification attributes.
10981	pthread_attr_t *	sigev_notify_attributes	

10982 The <signal.h> header shall define the following symbolic constants for the values of
 10983 *sigev_notify*:

10984 SIGEV_NONE No asynchronous notification is delivered when the event of interest
 10985 occurs.

10986 SIGEV_SIGNAL A queued signal, with an application-defined value, is generated when
 10987 the event of interest occurs.

10988 SIGEV_THREAD A notification function is called to perform notification.

Headers

<signal.h>

10989 The **sigval** union shall be defined as:

10990 int sival_int Integer signal value.

10991 void *sival_ptr Pointer signal value.

10992 The <**signal.h**> header shall declare the SIGRTMIN and SIGRTMAX macros, which shall expand

10993 to positive integer expressions with type **int**, but which need not be constant expressions. These

10994 macros specify a range of signal numbers that are reserved for application use and for which the

10995 realtime signal behavior specified in this volume of POSIX.1-2008 is supported. The signal

10996 numbers in this range do not overlap any of the signals specified in the following table.

10997 The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG_MAX} signal

10998 numbers.

10999 It is implementation-defined whether realtime signal behavior is supported for other signals.

11000 The <**signal.h**> header shall define the following macros that are used to refer to the signals that

11001 occur in the system. Signals defined here begin with the letters SIG followed by an uppercase

11002 letter. The macros shall expand to positive integer constant expressions with type **int** and

11003 distinct values. The value 0 is reserved for use as the null signal (see *kill()*). Additional

11004 implementation-defined signals may occur in the system.

11005 CX The ISO C standard only requires the signal names SIGABRT, SIGFPE, SIGILL, SIGINT,

11006 SIGSEGV, and SIGTERM to be defined.

11007 The following signals shall be supported on all implementations (default actions are explained

11008 below the table):

<signal.h>

Headers

	Signal	Default Action	Description
11009			
11010	SIGABRT	A	Process abort signal.
11011	SIGALRM	T	Alarm clock.
11012	SIGBUS	A	Access to an undefined portion of a memory object.
11013	SIGCHLD	I	Child process terminated, stopped,
11014	XSI		or continued.
11015	SIGCONT	C	Continue executing, if stopped.
11016	SIGFPE	A	Erroneous arithmetic operation.
11017	SIGHUP	T	Hangup.
11018	SIGILL	A	Illegal instruction.
11019	SIGINT	T	Terminal interrupt signal.
11020	SIGKILL	T	Kill (cannot be caught or ignored).
11021	SIGPIPE	T	Write on a pipe with no one to read it.
11022	SIGQUIT	A	Terminal quit signal.
11023	SIGSEGV	A	Invalid memory reference.
11024	SIGSTOP	S	Stop executing (cannot be caught or ignored).
11025	SIGTERM	T	Termination signal.
11026	SIGTSTP	S	Terminal stop signal.
11027	SIGTTIN	S	Background process attempting read.
11028	SIGTTOU	S	Background process attempting write.
11029	SIGUSR1	T	User-defined signal 1.
11030	SIGUSR2	T	User-defined signal 2.
11031	OB XSR	T	Pollable event.
11032	OB XSR	T	Profiling timer expired.
11033	XSI	A	Bad system call.
11034		A	Trace/breakpoint trap.
11035		I	High bandwidth data is available at a socket.
11036	XSI	T	Virtual timer expired.
11037		A	CPU time limit exceeded.
11038		A	File size limit exceeded.

11039 The default actions are as follows:

- 11040 T Abnormal termination of the process.
- 11041 XSI A Abnormal termination of the process with additional actions.
- 11042 I Ignore the signal.
- 11043 S Stop the process.
- 11044 C Continue the process, if it is stopped; otherwise, ignore the signal.

11045 The effects on the process in each case are described in XSH [Section 2.4.3](#) (on page 486).

11046 CX The <signal.h> header shall declare the **sigaction** structure, which shall include at least the

11047 following members:

11048	void (*sa_handler) (int)	Pointer to a signal-catching function
11049		or one of the SIG_IGN or SIG_DFL.
11050	sigset_t sa_mask	Set of signals to be blocked during execution
11051		of the signal handling function.
11052	int sa_flags	Special flags.
11053	void (*sa_sigaction) (int, siginfo_t *, void *)	
11054		Pointer to a signal-catching function.

Headers

<signal.h>

11055	XSI	The storage occupied by <i>sa_handler</i> and <i>sa_sigaction</i> may overlap, and a conforming application shall not use both simultaneously.	
11056			
11057		The <signal.h> header shall define the following macros which shall expand to integer constant expressions that need not be usable in #if preprocessing directives:	
11058			
11059	CX	SIG_BLOCK	The resulting set is the union of the current set and the signal set pointed to by the argument <i>set</i> .
11060			
11061	CX	SIG_UNBLOCK	The resulting set is the intersection of the current set and the complement of the signal set pointed to by the argument <i>set</i> .
11062			
11063	CX	SIG_SETMASK	The resulting set is the signal set pointed to by the argument <i>set</i> .
11064		The <signal.h> header shall also define the following symbolic constants:	
11065	CX	SA_NOCLDSTOP	Do not generate SIGCHLD when children stop or stopped children continue.
11066	XSI		
11067	XSI	SA_ONSTACK	Causes signal delivery to occur on an alternate stack.
11068	CX	SA_RESETHAND	Causes signal dispositions to be set to SIG_DFL on entry to signal handlers.
11069			
11070	CX	SA_RESTART	Causes certain functions to become restartable.
11071	CX	SA_SIGINFO	Causes extra information to be passed to signal handlers at the time of receipt of a signal.
11072			
11073	CX	SA_NOCLDWAIT	Causes implementations not to create zombie processes on child death.
11074	CX	SA_NODEFER	Causes signal not to be automatically blocked on entry to signal handler.
11075	XSI	SS_ONSTACK	Process is executing on an alternate signal stack.
11076	XSI	SS_DISABLE	Alternate signal stack is disabled.
11077	XSI	MINSIGSTKSZ	Minimum stack size for a signal handler.
11078	XSI	SIGSTKSZ	Default size in bytes for the alternate signal stack.
11079	CX	The <signal.h> header shall define the mcontext_t type through typedef .	
11080	CX	The <signal.h> header shall define the ucontext_t type as a structure that shall include at least the following members:	
11081			
11082		ucontext_t *uc_link	Pointer to the context that is resumed when this context returns.
11083			
11084		sigset_t uc_sigmask	The set of signals that are blocked when this context is active.
11085			
11086		stack_t uc_stack	The stack used by this context.
11087		mcontext_t uc_mcontext	A machine-specific representation of the saved context.
11088			
11089		The <signal.h> header shall define the stack_t type as a structure, which shall include at least the following members:	
11090			
11091		void *ss_sp	Stack base or pointer.
11092		size_t ss_size	Stack size.
11093		int ss_flags	Flags.

<signal.h>

Headers

11094	CX	The <signal.h> header shall define the siginfo_t type as a structure, which shall include at least the following members:		
11095				
11096	CX	int	si_signo	Signal number.
11097		int	si_code	Signal code.
11098	XSI	int	si_errno	If non-zero, an <i>errno</i> value associated with this signal, as described in <errno.h>.
11099				
11100	CX	pid_t	si_pid	Sending process ID.
11101		uid_t	si_uid	Real user ID of sending process.
11102		void	*si_addr	Address of faulting instruction.
11103		int	si_status	Exit value or signal.
11104	OB XSR	long	si_band	Band event for SIGPOLL.
11105	CX	union sigval	si_value	Signal value.
11106	CX	The <signal.h> header shall define the symbolic constants in the Code column of the following table for use as values of <i>si_code</i> that are signal-specific or non-signal-specific reasons why the signal was generated.		
11107				
11108				

Headers

<signal.h>

	Signal	Code	Reason
11109	CX	SIGILL	ILL_ILLOPC
11110			Illegal opcode.
11111		ILL_ILLOPN	Illegal operand.
11112		ILL_ILLADR	Illegal addressing mode.
11113		ILL_ILLTRP	Illegal trap.
11114		ILL_PRVOPC	Privileged opcode.
11115		ILL_PRVREG	Privileged register.
11116		ILL_COPROC	Coprocessor error.
11117		ILL_BADSTK	Internal stack error.
11118		SIGFPE	FPE_INTDIV
11119			Integer divide by zero.
11120		FPE_INTOVF	Integer overflow.
11121		FPE_FLTDIV	Floating-point divide by zero.
11122		FPE_FLTOVF	Floating-point overflow.
11123		FPE_FLTUND	Floating-point underflow.
11124		FPE_FLTRES	Floating-point inexact result.
11125		FPE_FLTINV	Invalid floating-point operation.
11126		SIGSEGV	SEGV_MAPERR
11127			Address not mapped to object.
11128			SEGV_ACCERR
11129			Invalid permissions for mapped object.
11130		SIGBUS	BUS_ADRALN
11131	XSI		Invalid address alignment.
11132			BUS_ADRERR
11133	CX		Nonexistent physical address.
11134			BUS_OBJERR
11135			Object-specific hardware error.
11136		SIGTRAP	TRAP_BRKPT
11137			Process breakpoint.
11138			TRAP_TRACE
11139			Process trace trap.
11140	OB XSR	SIGCHLD	CLD_EXITED
11141			Child has exited.
11142			CLD_KILLED
11143			Child has terminated abnormally and did not create a core file.
11144			CLD_DUMPED
11145			Child has terminated abnormally and created a core file.
11146			CLD_TRAPPED
11147	CX		Traced child has trapped.
11148			CLD_STOPPED
11149			Child has stopped.
11150			CLD_CONTINUED
11151			Stopped child has continued.
11152		SIGPOLL	POLL_IN
11153			Data input available.
11154			POLL_OUT
11155			Output buffers available.
11156			POLL_MSG
			Input message available.
			POLL_ERR
			I/O error.
			POLL_PRI
			High priority input available.
			POLL_HUP
			Device disconnected.
		Any	SI_USER
			Signal sent by <i>kill()</i> .
			SI_QUEUE
			Signal sent by the <i>sigqueue()</i> .
			SI_TIMER
			Signal generated by expiration of a timer set by <i>timer_settime()</i> .
			SI_ASYNCIO
			Signal generated by completion of an asynchronous I/O request.
			SI_MESGQ
			Signal generated by arrival of a message on an empty message queue
11152	CX	Implementations may support additional <i>si_code</i> values not included in this list, may generate values included in this list under circumstances other than those described in this list, and may contain extensions or limitations that prevent some values from being generated. Implementations do not generate a different value from the ones described in this list for circumstances described in this list.	
11153			
11154			
11155			
11156			

<signal.h>

Headers

11157 CX In addition, the following signal-specific information shall be available:

Signal	Member	Value
11158 11159 11160 SIGILL SIGFPE	void * <i>si_addr</i>	Address of faulting instruction.
11161 11162 SIGSEGV SIGBUS	void * <i>si_addr</i>	Address of faulting memory reference.
11163 11164 11165 SIGCHLD	pid_t <i>si_pid</i> int <i>si_status</i> uid_t <i>si_uid</i>	Child process ID. Exit value or signal. Real user ID of the process that sent the signal.
11166 OB XSR	SIGPOLL long <i>si_band</i>	Band event for POLL_IN, POLL_OUT, or POLL_MSG

11167 For some implementations, the value of *si_addr* may be inaccurate.

11168 The following shall be declared as functions and may also be defined as macros. Function
11169 prototypes shall be provided.

```

11170 CX  int    kill(pid_t, int);
11171 XSI  int    killpg(pid_t, int);
11172 CX  void    psiginfo(const siginfo_t *, const char *);
11173      void    psignal(int, const char *);
11174      int    pthread_kill(pthread_t, int);
11175      int    pthread_sigmask(int, const sigset_t *restrict,
11176                             sigset_t *restrict);
11177      int    raise(int);
11178 CX  int    sigaction(int, const struct sigaction *restrict,
11179                     struct sigaction *restrict);
11180      int    sigaddset(sigset_t *, int);
11181 XSI  int    sigaltstack(const stack_t *restrict, stack_t *restrict);
11182 CX  int    sigdelset(sigset_t *, int);
11183      int    sigemptyset(sigset_t *);
11184      int    sigfillset(sigset_t *);
11185 OB XSI int    sighold(int);
11186      int    sigignore(int);
11187      int    siginterrupt(int, int);
11188 CX  int    sigismember(const sigset_t *, int);
11189      void (*signal(int, void (*)(int)))(int);
11190 OB XSI int    sigpause(int);
11191 CX  int    sigpending(sigset_t *);
11192      int    sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
11193      int    sigqueue(pid_t, int, const union sigval);
11194 OB XSI int    sigrelse(int);
11195      void (*sigset(int, void (*)(int)))(int);
11196 CX  int    sigsuspend(const sigset_t *);
11197      int    sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
11198                          const struct timespec *restrict);
11199      int    sigwait(const sigset_t *restrict, int *restrict);
11200      int    sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);

```

11201 CX Inclusion of the <signal.h> header may make visible all symbols from the <time.h> header.

APPLICATION USAGE

On systems not supporting the XSI option, the *si_pid* and *si_uid* members of **siginfo_t** are only required to be valid when *si_code* is SI_USER or SI_QUEUE. On XSI-conforming systems, they are also valid for all *si_code* values less than or equal to 0; however, it is unspecified whether SI_USER and SI_QUEUE have values less than or equal to zero, and therefore XSI applications should check whether *si_code* has the value SI_USER or SI_QUEUE or is less than or equal to 0 to tell whether *si_pid* and *si_uid* are valid.

RATIONALE

None.

FUTURE DIRECTIONS

The SIGPOLL and SIGPROF signals may be removed in a future version.

SEE ALSO

<errno.h>, <stropts.h>, <sys/types.h>, <time.h>

XSH Section 2.2 (on page 468), *alarm()*, *ioctl()*, *kill()*, *killpg()*, *psiginfo()*, *pthread_kill()*, *pthread_sigmask()*, *raise()*, *sigaction()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sighold()*, *siginterrupt()*, *sigismember()*, *signal()*, *sigpending()*, *sigqueue()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, *timer_create()*, *wait()*, *waitid()*

CHANGE HISTORY

First released in Issue 1.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

The default action for SIGURG is changed from i to iii. The function prototype for *sigmask()* is removed.

Issue 6

The Open Group Corrigendum U035/2 is applied. In the DESCRIPTION, the wording for abnormal termination is clarified.

The Open Group Corrigendum U028/8 is applied, correcting the prototype for the *sigset()* function.

The Open Group Corrigendum U026/3 is applied, correcting the type of the *sigev_notify_function* function member of the **sigevent** structure.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now mandated. This is also a FIPS requirement.
- The **pid_t** definition is mandated.

The RT markings are changed to RTS to denote that the semantics are part of the Realtime Signals Extension option.

The **restrict** keyword is added to the prototypes for *sigaction()*, *sigaltstack()*, *sigprocmask()*, *sigtimedwait()*, *sigwait()*, and *sigwaitinfo()*.

IEEE PASC Interpretation 1003.1 #85 is applied, adding the statement that symbols from <time.h> may be made visible when <signal.h> is included.

Extensions beyond the ISO C standard are marked.

<signal.h>

Headers

- 11245 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/14 is applied, changing the descriptive
11246 text for members of the **sigaction** structure.
- 11247 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/15 is applied, correcting the definition of
11248 the *sa_sigaction* member of the **sigaction** structure.
- 11249 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/24 is applied, reworking the ordering of
11250 the **siginfo_t** type structure in the DESCRIPTION. This is an editorial change and no normative
11251 change is intended.
- 11252 **Issue 7**
- 11253 SD5-XBD-ERN-5 is applied.
- 11254 SD5-XBD-ERN-39 is applied, removing the **sigstack** structure which should have been removed
11255 at the same time as the LEGACY *sigstack()* function.
- 11256 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size_t** type.
- 11257 Austin Group Interpretation 1003.1-2001 #034 is applied.
- 11258 The **ucontext_t** and **mcontext_t** structures are added here from the obsolescent <ucontext.h>
11259 header.
- 11260 The *psiginfo()* and *psignal()* functions are added from The Open Group Technical Standard, 2006,
11261 Extended API Set Part 1.
- 11262 The SIGPOLL and SIGPROF signals and text relating to the XSI STREAMS option are marked
11263 obsolescent.
- 11264 The SA_RESETHAND, SA_RESTART, SA_SIGINFO, SA_NOCLDWAIT, and SA_NODEFER
11265 constants are moved from the XSI option to the Base.
- 11266 Functionality relating to the Realtime Signals Extension option is moved to the Base.
- 11267 This reference page is clarified with respect to macros and symbolic constants, and declarations
11268 for the **pthread_attr_t**, **pthread_t**, and **uid_t** types and the **timespec** structure are added.
- 11269 SIGRTMIN and SIGRTMAX are required to be positive integer expressions.
- 11270 The APPLICATION USAGE section is updated to describe the *si_pid* and *si_uid* members of
11271 **siginfo_t**.

Headers

<spawn.h>

11272 NAME

11273 spawn.h — spawn (ADVANCED REALTIME)

11274 SYNOPSIS

11275 SPN #include <spawn.h>

11276 DESCRIPTION

11277 The <spawn.h> header shall define the **posix_spawnattr_t** and **posix_spawn_file_actions_t**
 11278 types used in performing spawn operations.

11279 The <spawn.h> header shall define the **mode_t** and **pid_t** types as described in <sys/types.h>.

11280 The <spawn.h> header shall define the **sigset_t** type as described in <signal.h>.

11281 The tag **sched_param** shall be declared as naming an incomplete structure type, the contents of
 11282 which are described in the <sched.h> header.

11283 The <spawn.h> header shall define the following symbolic constants for use as the flags that
 11284 may be set in a **posix_spawnattr_t** object using the *posix_spawnattr_setflags()* function:

11285 POSIX_SPAWN_RESETIDS
 11286 POSIX_SPAWN_SETPGROUP
 11287 PS POSIX_SPAWN_SETSCHEDPARAM
 11288 POSIX_SPAWN_SETSCHEDULER
 11289 POSIX_SPAWN_SETSIGDEF
 11290 POSIX_SPAWN_SETSIGMASK

11291 The following shall be declared as functions and may also be defined as macros. Function
 11292 prototypes shall be provided.

```

11293 int    posix_spawn(pid_t *restrict, const char *restrict,
11294                  const posix_spawn_file_actions_t *,
11295                  const posix_spawnattr_t *restrict, char *const [restrict],
11296                  char *const [restrict]);
11297 int    posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
11298                  int);
11299 int    posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
11300                  int, int);
11301 int    posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict,
11302                  int, const char *restrict, int, mode_t);
11303 int    posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
11304 int    posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
11305 int    posix_spawnattr_destroy(posix_spawnattr_t *);
11306 int    posix_spawnattr_getflags(const posix_spawnattr_t *restrict,
11307                  short *restrict);
11308 int    posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
11309                  pid_t *restrict);
11310 PS int    posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
11311                  struct sched_param *restrict);
11312 int    posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
11313                  int *restrict);
11314 int    posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
11315                  sigset_t *restrict);
11316 int    posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
11317                  sigset_t *restrict);
11318 int    posix_spawnattr_init(posix_spawnattr_t *);

```

<spawn.h>

Headers

```

11319     int    posix_spawnattr_setflags(posix_spawnattr_t *, short);
11320     int    posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);
11321 PS     int    posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,
11322         const struct sched_param *restrict);
11323     int    posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);
11324     int    posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict,
11325         const sigset_t *restrict);
11326     int    posix_spawnattr_setsigmask(posix_spawnattr_t *restrict,
11327         const sigset_t *restrict);
11328     int    posix_spawn(pid_t *restrict, const char *restrict,
11329         const posix_spawn_file_actions_t *,
11330         const posix_spawnattr_t *restrict,
11331         char *const [restrict], char *const [restrict]);

```

11332 Inclusion of the <spawn.h> header may make visible symbols defined in the <sched.h> and
 11333 <signal.h> headers.

11334 APPLICATION USAGE

11335 None.

11336 RATIONALE

11337 None.

11338 FUTURE DIRECTIONS

11339 None.

11340 SEE ALSO

11341 <sched.h>, <semaphore.h>, <signal.h>, <sys/types.h>

11342 XSH *posix_spawn()*, *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*,
 11343 *posix_spawn_file_actions_destroy()*, *posix_spawnattr_destroy()*, *posix_spawnattr_getflags()*,
 11344 *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*,
 11345 *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getsigmask()*

11346 CHANGE HISTORY

11347 First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

11348 The **restrict** keyword is added to the prototypes for *posix_spawn()*,
 11349 *posix_spawn_file_actions_addopen()*, *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getflags()*,
 11350 *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*,
 11351 *posix_spawnattr_getsigmask()*, *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setschedparam()*,
 11352 *posix_spawnattr_setsigmask()*, and *posix_spawnnp()*.

11353 Issue 7

11354 This reference page is clarified with respect to macros and symbolic constants, and declarations
 11355 for the **mode_t**, **pid_t**, and **sigset_t** types are added.

Headers

<stdarg.h>

NAME

stdarg.h — handle variable argument list

SYNOPSIS

```
#include <stdarg.h>

void va_start(va_list ap, argN);
void va_copy(va_list dest, va_list src);
type va_arg(va_list ap, type);
void va_end(va_list ap);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The **<stdarg.h>** header shall contain a set of macros which allows portable functions that accept variable argument lists to be written. Functions that have variable argument lists (such as *printf()*) but do not use these macros are inherently non-portable, as different systems use different argument-passing conventions.

The **<stdarg.h>** header shall define the **va_list** type for variables used to traverse the list.

The *va_start()* macro is invoked to initialize *ap* to the beginning of the list before any calls to *va_arg()*.

The *va_copy()* macro initializes *dest* as a copy of *src*, as if the *va_start()* macro had been applied to *dest* followed by the same sequence of uses of the *va_arg()* macro as had previously been used to reach the present state of *src*. Neither the *va_copy()* nor *va_start()* macro shall be invoked to reinitialize *dest* without an intervening invocation of the *va_end()* macro for the same *dest*.

The object *ap* may be passed as an argument to another function; if that function invokes the *va_arg()* macro with parameter *ap*, the value of *ap* in the calling function is unspecified and shall be passed to the *va_end()* macro prior to any further reference to *ap*. The parameter *argN* is the identifier of the rightmost parameter in the variable parameter list in the function definition (the one just before the ...). If the parameter *argN* is declared with the **register** storage class, with a function type or array type, or with a type that is not compatible with the type that results after application of the default argument promotions, the behavior is undefined.

The *va_arg()* macro shall return the next argument in the list pointed to by *ap*. Each invocation of *va_arg()* modifies *ap* so that the values of successive arguments are returned in turn. The *type* parameter shall be a type name specified such that the type of a pointer to an object that has the specified type can be obtained simply by postfixing a '*' to *type*. If there is no actual next argument, or if *type* is not compatible with the type of the actual next argument (as promoted according to the default argument promotions), the behavior is undefined, except for the following cases:

- One type is a signed integer type, the other type is the corresponding unsigned integer type, and the value is representable in both types.
- One type is a pointer to **void** and the other is a pointer to a character type.
- Both types are pointers.

Different types can be mixed, but it is up to the routine to know what type of argument is expected.

The *va_end()* macro is used to clean up; it invalidates *ap* for use (unless *va_start()* or *va_copy()* is invoked again).

<stdarg.h>

Headers

Each invocation of the *va_start()* and *va_copy()* macros shall be matched by a corresponding invocation of the *va_end()* macro in the same function.

Multiple traversals, each bracketed by *va_start()* ... *va_end()*, are possible.

EXAMPLES

This example is a possible implementation of *execl()*:

```
#include <stdarg.h>

#define MAXARGS 31

/*
 * execl is called by
 * execl(file, arg1, arg2, ..., (char *) (0));
 */
int execl(const char *file, const char *args, ...)
{
    va_list ap;
    char *array[MAXARGS + 1];
    int argno = 0;

    va_start(ap, args);
    while (args != 0 && argno < MAXARGS)
    {
        array[argno++] = args;
        args = va_arg(ap, const char *);
    }
    array[argno] = (char *) 0;
    va_end(ap);
    return execv(file, array);
}
```

APPLICATION USAGE

It is up to the calling routine to communicate to the called routine how many arguments there are, since it is not always possible for the called routine to determine this in any other way. For example, *execl()* is passed a null pointer to signal the end of the list. The *printf()* function can tell how many arguments are there by the *format* argument.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *exec*, *fprintf()*

CHANGE HISTORY

First released in Issue 4. Derived from the ANSI C standard.

Issue 6

This reference page is updated to align with the ISO/IEC 9899:1999 standard.

Headers

<stdbool.h>

11442 **NAME**

11443 stdbool.h — boolean type and values

11444 **SYNOPSIS**

11445 #include <stdbool.h>

11446 **DESCRIPTION**

11447 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11448 conflict between the requirements described here and the ISO C standard is unintentional. This
 11449 volume of POSIX.1-2008 defers to the ISO C standard.

11450 The <stdbool.h> header shall define the following macros:

11451 bool Expands to **_Bool**.

11452 true Expands to the integer constant 1.

11453 false Expands to the integer constant 0.

11454 __bool_true_false_are_defined

11455 Expands to the integer constant 1.

11456 An application may undefine and then possibly redefine the macros bool, true, and false.

11457 **APPLICATION USAGE**

11458 None.

11459 **RATIONALE**

11460 None.

11461 **FUTURE DIRECTIONS**

11462 The ability to undefine and redefine the macros bool, true, and false is an obsolescent feature
 11463 and may be removed in a future version.

11464 **SEE ALSO**

11465 None.

11466 **CHANGE HISTORY**

11467 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

<stddef.h>

Headers

11468 NAME

11469 stddef.h — standard type definitions

11470 SYNOPSIS

11471 #include <stddef.h>

11472 DESCRIPTION

11473 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11474 conflict between the requirements described here and the ISO C standard is unintentional. This
11475 volume of POSIX.1-2008 defers to the ISO C standard.

11476 The <stddef.h> header shall define the following macros:

11477 CX NULL Null pointer constant. The macro shall expand to an integer constant expression
11478 with the value 0 cast to type **void ***.

11479 offsetof(*type*, *member-designator*)

11480 Integer constant expression of type **size_t**, the value of which is the offset in bytes
11481 to the structure member (*member-designator*), from the beginning of its structure
11482 (*type*).

11483 The <stddef.h> header shall define the following types:

11484 **ptrdiff_t** Signed integer type of the result of subtracting two pointers.

11485 **wchar_t** Integer type whose range of values can represent distinct codes for all members of
11486 the largest extended character set specified among the supported locales; the null
11487 character shall have the code value zero. Each member of the basic character set
11488 shall have a code value equal to its value when used as the lone character in an
11489 integer character constant if an implementation does not define
11490 __STDC_MB_MIGHT_NEQ_WC__.

11491 **size_t** Unsigned integer type of the result of the *sizeof* operator.

11492 The implementation shall support one or more programming environments in which the widths
11493 of **ptrdiff_t**, **size_t**, and **wchar_t** are no greater than the width of type **long**. The names of these
11494 programming environments can be obtained using the *confstr()* function or the *getconf* utility.

11495 APPLICATION USAGE

11496 None.

11497 RATIONALE

11498 The ISO C standard does not require the NULL macro to include the cast to type **void *** and
11499 specifies that the NULL macro be implementation-defined. POSIX.1-2008 requires the cast and
11500 therefore need not be implementation-defined.

11501 FUTURE DIRECTIONS

11502 None.

11503 SEE ALSO

11504 <sys/types.h>, <wchar.h>

11505 XSH *confstr()*

11506 XCU *getconf*

11507 CHANGE HISTORY

11508 First released in Issue 4. Derived from the ANSI C standard.

Headers

<stddef.h>

11509 **Issue 7**

11510 This reference page is clarified with respect to macros and symbolic constants.

11511 SD5-XBD-ERN-53 is applied, updating the definition of `wchar_t` to align with
11512 ISO/IEC 9899:1999 standard, Technical Corrigendum 3.

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

<stdint.h>

Headers

11513 NAME

11514 stdint.h — integer types

11515 SYNOPSIS

11516 #include <stdint.h>

11517 DESCRIPTION

11518 CX Some of the functionality described on this reference page extends the ISO C standard.
 11519 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to
 11520 enable the visibility of these symbols in this header.

11521 The <stdint.h> header shall declare sets of integer types having specified widths, and shall
 11522 define corresponding sets of macros. It shall also define macros that specify limits of integer
 11523 types corresponding to types defined in other standard headers.

11524 **Note:** The “width” of an integer type is the number of bits used to store its value in a pure binary
 11525 system; the actual type may use more bits than that (for example, a 28-bit type could be stored
 11526 in 32 bits of actual storage). An N -bit signed type has values in the range -2^{N-1} or $1-2^{N-1}$ to
 11527 $2^{N-1}-1$, while an N -bit unsigned type has values in the range 0 to 2^N-1 .

11528 Types are defined in the following categories:

- 11529 • Integer types having certain exact widths
- 11530 • Integer types having at least certain specified widths
- 11531 • Fastest integer types having at least certain specified widths
- 11532 • Integer types wide enough to hold pointers to objects
- 11533 • Integer types having greatest width

11534 (Some of these types may denote the same type.)

11535 Corresponding macros specify limits of the declared types and construct suitable constants.

11536 For each type described herein that the implementation provides, the <stdint.h> header shall
 11537 declare that **typedef** name and define the associated macros. Conversely, for each type described
 11538 herein that the implementation does not provide, the <stdint.h> header shall not declare that
 11539 **typedef** name, nor shall it define the associated macros. An implementation shall provide those
 11540 types described as required, but need not provide any of the others (described as optional).

11541 Integer Types

11542 When **typedef** names differing only in the absence or presence of the initial u are defined, they
 11543 shall denote corresponding signed and unsigned types as described in the ISO/IEC 9899:1999
 11544 standard Section 6.2.5; an implementation providing one of these corresponding types shall also
 11545 provide the other.

11546 In the following descriptions, the symbol N represents an unsigned decimal integer with no
 11547 leading zeros (for example, 8 or 24, but not 04 or 048).

- 11548 • Exact-width integer types

11549 The **typedef** name **int N _t** designates a signed integer type with width N , no padding bits,
 11550 and a two’s-complement representation. Thus, **int8_t** denotes a signed integer type with a
 11551 width of exactly 8 bits.

11552 The **typedef** name **uint N _t** designates an unsigned integer type with width N . Thus,
 11553 **uint24_t** denotes an unsigned integer type with a width of exactly 24 bits.

Headers

<stdint.h>

11554 CX

The following types are required:

11555 **int8_t**
 11556 **int16_t**
 11557 **int32_t**
 11558 **uint8_t**
 11559 **uint16_t**
 11560 **uint32_t**

11561 If an implementation provides integer types with width 64 that meet these requirements,
 11562 then the following types are required:

11563 **int64_t**
 11564 **uint64_t**

11565 CX

In particular, this will be the case if any of the following are true:

- 11566 — The implementation supports the `_POSIX_V7_ILP32_OFFBIG` programming
 11567 environment and the application is being built in the `_POSIX_V7_ILP32_OFFBIG`
 11568 programming environment (see the Shell and Utilities volume of POSIX.1-2008, *c99*,
 11569 Programming Environments).
- 11570 — The implementation supports the `_POSIX_V7_LP64_OFF64` programming
 11571 environment and the application is being built in the `_POSIX_V7_LP64_OFF64`
 11572 programming environment.
- 11573 — The implementation supports the `_POSIX_V7_LPBIG_OFFBIG` programming
 11574 environment and the application is being built in the `_POSIX_V7_LPBIG_OFFBIG`
 11575 programming environment.

11576 All other types of this form are optional.

11577 • Minimum-width integer types

11578 The **typedef** name **int_leastN_t** designates a signed integer type with a width of at least *N*,
 11579 such that no signed integer type with lesser size has at least the specified width. Thus,
 11580 **int_least32_t** denotes a signed integer type with a width of at least 32 bits.

11581 The **typedef** name **uint_leastN_t** designates an unsigned integer type with a width of at
 11582 least *N*, such that no unsigned integer type with lesser size has at least the specified width.
 11583 Thus, **uint_least16_t** denotes an unsigned integer type with a width of at least 16 bits.

11584 The following types are required:

11585 **int_least8_t**
 11586 **int_least16_t**
 11587 **int_least32_t**
 11588 **int_least64_t**
 11589 **uint_least8_t**
 11590 **uint_least16_t**
 11591 **uint_least32_t**
 11592 **uint_least64_t**

11593 All other types of this form are optional.

- 11594 • Fastest minimum-width integer types
- 11595 Each of the following types designates an integer type that is usually fastest to operate
11596 with among all integer types that have at least the specified width.
- 11597 The designated type is not guaranteed to be fastest for all purposes; if the implementation
11598 has no clear grounds for choosing one type over another, it will simply pick some integer
11599 type satisfying the signedness and width requirements.
- 11600 The **typedef** name **int_fastN_t** designates the fastest signed integer type with a width of at
11601 least *N*. The **typedef** name **uint_fastN_t** designates the fastest unsigned integer type with
11602 a width of at least *N*.
- 11603 The following types are required:
- 11604 **int_fast8_t**
11605 **int_fast16_t**
11606 **int_fast32_t**
11607 **int_fast64_t**
11608 **uint_fast8_t**
11609 **uint_fast16_t**
11610 **uint_fast32_t**
11611 **uint_fast64_t**
- 11612 All other types of this form are optional.
- 11613 • Integer types capable of holding object pointers
- 11614 The following type designates a signed integer type with the property that any valid
11615 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and
11616 the result will compare equal to the original pointer:
- 11617 **intptr_t**
- 11618 The following type designates an unsigned integer type with the property that any valid
11619 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and
11620 the result will compare equal to the original pointer:
- 11621 **uintptr_t**
- 11622 XSI On XSI-conformant systems, the **intptr_t** and **uintptr_t** types are required; otherwise, they
11623 are optional.
- 11624 • Greatest-width integer types
- 11625 The following type designates a signed integer type capable of representing any value of
11626 any signed integer type:
- 11627 **intmax_t**
- 11628 The following type designates an unsigned integer type capable of representing any value
11629 of any unsigned integer type:
- 11630 **uintmax_t**
- 11631 These types are required.

Note: Applications can test for optional types by using the corresponding limit macro from [Limits of Specified-Width Integer Types](#).

Limits of Specified-Width Integer Types

The following macros specify the minimum and maximum limits of the types declared in the <stdint.h> header. Each macro name corresponds to a similar type name in [Integer Types](#) (on page 344).

Each instance of any defined macro shall be replaced by a constant expression suitable for use in **#if** preprocessing directives, and this expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. Its implementation-defined value shall be equal to or greater in magnitude (absolute value) than the corresponding value given below, with the same sign, except where stated to be exactly the given value.

- Limits of exact-width integer types

- Minimum values of exact-width signed integer types:

{INTN_MIN} Exactly $-(2^{N-1})$

- Maximum values of exact-width signed integer types:

{INTN_MAX} Exactly $2^{N-1} - 1$

- Maximum values of exact-width unsigned integer types:

{UINTN_MAX} Exactly $2^N - 1$

- Limits of minimum-width integer types

- Minimum values of minimum-width signed integer types:

{INT_LEASTN_MIN} $-(2^{N-1} - 1)$

- Maximum values of minimum-width signed integer types:

{INT_LEASTN_MAX} $2^{N-1} - 1$

- Maximum values of minimum-width unsigned integer types:

{UINT_LEASTN_MAX} $2^N - 1$

- Limits of fastest minimum-width integer types

- Minimum values of fastest minimum-width signed integer types:

{INT_FASTN_MIN} $-(2^{N-1} - 1)$

- Maximum values of fastest minimum-width signed integer types:

{INT_FASTN_MAX} $2^{N-1} - 1$

- Maximum values of fastest minimum-width unsigned integer types:

{UINT_FASTN_MAX} $2^N - 1$

- Limits of integer types capable of holding object pointers

- Minimum value of pointer-holding signed integer type:

{INTPTR_MIN} $-(2^{15} - 1)$

<stdint.h>

Headers

- 11668 — Maximum value of pointer-holding signed integer type:
- 11669 {INTPTR_MAX} $2^{15} - 1$
- 11670 — Maximum value of pointer-holding unsigned integer type:
- 11671 {UINTPTR_MAX} $2^{16} - 1$
- 11672 • Limits of greatest-width integer types
- 11673 — Minimum value of greatest-width signed integer type:
- 11674 {INTMAX_MIN} $-(2^{63} - 1)$
- 11675 — Maximum value of greatest-width signed integer type:
- 11676 {INTMAX_MAX} $2^{63} - 1$
- 11677 — Maximum value of greatest-width unsigned integer type:
- 11678 {UINTMAX_MAX} $2^{64} - 1$

Limits of Other Integer Types

The following macros specify the minimum and maximum limits of integer types corresponding to types defined in other standard headers.

Each instance of these macros shall be replaced by a constant expression suitable for use in **#if** preprocessing directives, and this expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. Its implementation-defined value shall be equal to or greater in magnitude (absolute value) than the corresponding value given below, with the same sign.

- 11687 • Limits of **ptrdiff_t**:
- 11688 {PTRDIFF_MIN} $-65\,535$
- 11689 {PTRDIFF_MAX} $+65\,535$
- 11690 • Limits of **sig_atomic_t**:
- 11691 {SIG_ATOMIC_MIN} See below.
- 11692 {SIG_ATOMIC_MAX} See below.
- 11693 • Limit of **size_t**:
- 11694 {SIZE_MAX} $65\,535$
- 11695 • Limits of **wchar_t**:
- 11696 {WCHAR_MIN} See below.
- 11697 {WCHAR_MAX} See below.
- 11698 • Limits of **wint_t**:
- 11699 {WINT_MIN} See below.
- 11700 {WINT_MAX} See below.

If **sig_atomic_t** (see the <signal.h> header) is defined as a signed integer type, the value of {SIG_ATOMIC_MIN} shall be no greater than -127 and the value of {SIG_ATOMIC_MAX} shall be no less than 127 ; otherwise, **sig_atomic_t** shall be defined as an unsigned integer type, and the value of {SIG_ATOMIC_MIN} shall be 0 and the value of {SIG_ATOMIC_MAX} shall be no

Headers

<stdint.h>

less than 255.

If **wchar_t** (see the <stddef.h> header) is defined as a signed integer type, the value of {WCHAR_MIN} shall be no greater than -127 and the value of {WCHAR_MAX} shall be no less than 127; otherwise, **wchar_t** shall be defined as an unsigned integer type, and the value of {WCHAR_MIN} shall be 0 and the value of {WCHAR_MAX} shall be no less than 255.

If **wint_t** (see the <wchar.h> header) is defined as a signed integer type, the value of {WINT_MIN} shall be no greater than -32 767 and the value of {WINT_MAX} shall be no less than 32 767; otherwise, **wint_t** shall be defined as an unsigned integer type, and the value of {WINT_MIN} shall be 0 and the value of {WINT_MAX} shall be no less than 65 535.

Macros for Integer Constant Expressions

The following macros expand to integer constant expressions suitable for initializing objects that have integer types corresponding to types defined in the <stdint.h> header. Each macro name corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-width integer types*.

Each invocation of one of these macros shall expand to an integer constant expression suitable for use in **#if** preprocessing directives. The type of the expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. The value of the expression shall be that of the argument.

The argument in any instance of these macros shall be an unsuffixed integer constant with a value that does not exceed the limits for the corresponding type.

- Macros for minimum-width integer constant expressions

The macro **INTN_C(value)** shall expand to an integer constant expression corresponding to the type **int_leastN_t**. The macro **UINTN_C(value)** shall expand to an integer constant expression corresponding to the type **uint_leastN_t**. For example, if **uint_least64_t** is a name for the type **unsigned long long**, then **UINT64_C(0x123)** might expand to the integer constant **0x123ULL**.

- Macros for greatest-width integer constant expressions

The following macro expands to an integer constant expression having the value specified by its argument and the type **intmax_t**:

INTMAX_C(value)

The following macro expands to an integer constant expression having the value specified by its argument and the type **uintmax_t**:

UINTMAX_C(value)

APPLICATION USAGE

None.

RATIONALE

The <stdint.h> header is a subset of the <inttypes.h> header more suitable for use in freestanding environments, which might not support the formatted I/O functions. In some environments, if the formatted conversion support is not wanted, using this header instead of the <inttypes.h> header avoids defining such a large number of macros.

<stdint.h>

Headers

- 11745 As a consequence of adding **int8_t**, the following are true:
- 11746 • A byte is exactly 8 bits.
- 11747 • {CHAR_BIT} has the value 8, {SCHAR_MAX} has the value 127, {SCHAR_MIN} has the
- 11748 value -128, and {UCHAR_MAX} has the value 255.
- 11749 (The POSIX standard explicitly requires 8-bit char and two's-complement arithmetic.)
- 11750 **FUTURE DIRECTIONS**
- 11751 **typedef** names beginning with **int** or **uint** and ending with **_t** may be added to the types defined
- 11752 in the **<stdint.h>** header. Macro names beginning with **INT** or **UINT** and ending with **_MAX**,
- 11753 **_MIN**, or **_C** may be added to the macros defined in the **<stdint.h>** header.
- 11754 **SEE ALSO**
- 11755 [**<inttypes.h>**](#), [**<signal.h>**](#), [**<stddef.h>**](#), [**<wchar.h>**](#)
- 11756 XSH [Section 2.2](#) (on page 468)
- 11757 **CHANGE HISTORY**
- 11758 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.
- 11759 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is applied.
- 11760 **Issue 7**
- 11761 ISO/IEC 9899:1999 standard, Technical Corrigendum 3 #40 is applied.
- 11762 SD5-XBD-ERN-67 is applied.

Headers

<stdio.h>

11763 NAME

11764 stdio.h — standard buffered input/output

11765 SYNOPSIS

11766 #include <stdio.h>

11767 DESCRIPTION

11768 CX Some of the functionality described on this reference page extends the ISO C standard.
 11769 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to
 11770 enable the visibility of these symbols in this header.

11771 The <stdio.h> header shall define the following data types through **typedef**:11772 **FILE** A structure containing information about a file.

11773 **fpos_t** A non-array type containing all information needed to specify uniquely
 11774 every position within a file.

11775 **off_t** As described in <sys/types.h>.11776 **size_t** As described in <stddef.h>.11777 CX **ssize_t** As described in <sys/types.h>.11778 CX **va_list** As described in <stdarg.h>.

11779 The <stdio.h> header shall define the following macros which shall expand to integer constant
 11780 expressions:

11781 CX **BUFSIZ** Size of <stdio.h> buffers. This shall expand to a positive value.11782 CX **L_ctermid** Maximum size of character array to hold *ctermid()* output.11783 OB **L_tmpnam** Maximum size of character array to hold *tmpnam()* output.

11784 The <stdio.h> header shall define the following macros which shall expand to integer constant
 11785 expressions with distinct values:

11786 **_IOFBF** Input/output fully buffered.11787 **_IOLBF** Input/output line buffered.11788 **_IONBF** Input/output unbuffered.

11789 The <stdio.h> header shall define the following macros which shall expand to integer constant
 11790 expressions with distinct values:

11791 **SEEK_CUR** Seek relative to current position.11792 **SEEK_END** Seek relative to end-of-file.11793 **SEEK_SET** Seek relative to start-of-file.

11794 The <stdio.h> header shall define the following macros which shall expand to integer constant
 11795 expressions denoting implementation limits:

11796 {FILENAME_MAX} Maximum size in bytes of the longest filename string that the
 11797 implementation guarantees can be opened.

11798 {FOPEN_MAX} Number of streams which the implementation guarantees can be open
 11799 simultaneously. The value is at least eight.

11800 OB {TMP_MAX} Minimum number of unique filenames generated by *tmpnam()*.
 11801 Maximum number of times an application can call *tmpnam()* reliably. The
 11802 value of {TMP_MAX} is at least 25.

<stdio.h>

Headers

11803	OB XSI	On XSI-conformant systems, the value of {TMP_MAX} is at least 10 000.
11804		The <stdio.h> header shall define the following macro which shall expand to an integer constant expression with type int and a negative value:
11805		
11806		EOF End-of-file return value.
11807		The <stdio.h> header shall define NULL as described in <stddef.h>.
11808		The <stdio.h> header shall define the following macro which shall expand to a string constant:
11809	OB XSI	P_tmpdir Default directory prefix for <i>tempnam()</i> .
11810		The <stdio.h> header shall define the following macros which shall expand to expressions of type “pointer to FILE ” that point to the FILE objects associated, respectively, with the standard error, input, and output streams:
11811		
11812		
11813		stderr Standard error output stream.
11814		stdin Standard input stream.
11815		stdout Standard output stream.
11816		The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.
11817		
11818		void clearerr(FILE *);
11819	CX	char *ctermid(char *);
11820		int dprintf(int, const char *restrict, ...)
11821		int fclose(FILE *);
11822	CX	FILE *fdopen(int, const char *);
11823		int feof(FILE *);
11824		int ferror(FILE *);
11825		int fflush(FILE *);
11826		int fgetc(FILE *);
11827		int fgetpos(FILE *restrict, fpos_t *restrict);
11828		char *fgets(char *restrict, int, FILE *restrict);
11829	CX	int fileno(FILE *);
11830		void flockfile(FILE *);
11831		FILE *fmemopen(void *restrict, size_t, const char *restrict);
11832		FILE *fopen(const char *restrict, const char *restrict);
11833		int fprintf(FILE *restrict, const char *restrict, ...);
11834		int fputc(int, FILE *);
11835		int fputs(const char *restrict, FILE *restrict);
11836		size_t fread(void *restrict, size_t, size_t, FILE *restrict);
11837		FILE *freopen(const char *restrict, const char *restrict, FILE *restrict);
11838		
11839		int fscanf(FILE *restrict, const char *restrict, ...);
11840		int fseek(FILE *, long, int);
11841	CX	int fseeko(FILE *, off_t, int);
11842		int fsetpos(FILE *, const fpos_t *);
11843		long ftell(FILE *);
11844	CX	off_t ftello(FILE *);
11845		int ftrylockfile(FILE *);
11846		void funlockfile(FILE *);
11847		size_t fwrite(const void *restrict, size_t, size_t, FILE *restrict);
11848		int getc(FILE *);

Headers

<stdio.h>

11849 int getchar(void);

11850 CX int getc_unlocked(FILE *);

11851 int getchar_unlocked(void);

11852 ssize_t getdelim(char **restrict, size_t *restrict, int,

11853 FILE *restrict);

11854 ssize_t getline(char **restrict, size_t *restrict, FILE *restrict);

11855 OB char *gets(char *);

11856 CX FILE *open_memstream(char **, size_t *);

11857 int pclose(FILE *);

11858 void perror(const char *);

11859 CX FILE *popen(const char *, const char *);

11860 int printf(const char *restrict, ...);

11861 int putc(int, FILE *);

11862 int putchar(int);

11863 CX int putc_unlocked(int, FILE *);

11864 int putchar_unlocked(int);

11865 int puts(const char *);

11866 int remove(const char *);

11867 int rename(const char *, const char *);

11868 CX int renameat(int, const char *, int, const char *);

11869 void rewind(FILE *);

11870 int scanf(const char *restrict, ...);

11871 void setbuf(FILE *restrict, char *restrict);

11872 int setvbuf(FILE *restrict, char *restrict, int, size_t);

11873 int snprintf(char *restrict, size_t, const char *restrict, ...);

11874 int sprintf(char *restrict, const char *restrict, ...);

11875 int sscanf(const char *restrict, const char *restrict, ...);

11876 OB XSI char *tempnam(const char *, const char *);

11877 FILE *tmpfile(void);

11878 OB char *tmpnam(char *);

11879 int ungetc(int, FILE *);

11880 CX int vdprintf(int, const char *restrict, va_list);

11881 int vfprintf(FILE *restrict, const char *restrict, va_list);

11882 int vfscanf(FILE *restrict, const char *restrict, va_list);

11883 int vprintf(const char *restrict, va_list);

11884 int vscanf(const char *restrict, va_list);

11885 int vsnprintf(char *restrict, size_t, const char *restrict,

11886 va_list);

11887 int vsprintf(char *restrict, const char *restrict, va_list);

11888 int vsscanf(const char *restrict, const char *restrict, va_list);

11889 CX Inclusion of the <stdio.h> header may also make visible all symbols from <stddef.h>.

APPLICATION USAGE

Since standard I/O streams may use an underlying file descriptor to access the file associated with a stream, application developers need to be aware that {FOPEN_MAX} streams may not be available if file descriptors are being used to access files that are not associated with streams.

RATIONALE

There is a conflict between the ISO C standard and the POSIX definition of the {TMP_MAX} macro that is addressed by ISO/IEC 9899:1999 standard, Defect Report 336. The POSIX standard is in alignment with the public record of the response to the Defect Report. This change has not yet been published as part of the ISO C standard.

<stdio.h>

Headers

11899 FUTURE DIRECTIONS

11900 None.

11901 SEE ALSO

11902 <stdarg.h>, <stddef.h>, <sys/types.h>

11903 XSH Section 2.2 (on page 468), *clearerr()*, *ctermid()*, *fclose()*, *fdopen()*, *feof()*, *ferror()*, *fflush()*,
 11904 *fgetc()*, *fgetpos()*, *fgets()*, *fileno()*, *flockfile()*, *fmemopen()*, *fopen()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*,
 11905 *freopen()*, *fscanf()*, *fseek()*, *fsetpos()*, *ftell()*, *fwrite()*, *getc()*, *getchar()*, *getc_unlocked()*, *getdelim()*,
 11906 *getopt()*, *gets()*, *open_memstream()*, *pclose()*, *perror()*, *popen()*, *putc()*, *putchar()*, *puts()*, *remove()*,
 11907 *rename()*, *rewind()*, *setbuf()*, *setvbuf()*, *stdin*, *system()*, *tempnam()*, *tmpfile()*, *tmpnam()*, *ungetc()*,
 11908 *vfprintf()*, *vfscanf()*

11909 CHANGE HISTORY

11910 First released in Issue 1. Derived from Issue 1 of the SVID.

11911 Issue 5

11912 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11913 Large File System extensions are added.

11914 The constant *L_cuserid* and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked as
 11915 extensions and LEGACY.

11916 The *cuserid()* and *getopt()* functions are marked LEGACY.

11917 Issue 6

11918 The constant *L_cuserid* and the external variables *optarg*, *opterr*, *optind*, and *optopt* are removed
 11919 as they were previously marked LEGACY.

11920 The *cuserid()*, *getopt()*, and *getw()* functions are removed as they were previously marked
 11921 LEGACY.

11922 Several functions are marked as part of the Thread-Safe Functions option.

11923 This reference page is updated to align with the ISO/IEC 9899:1999 standard. Note that the
 11924 description of the **fpos_t** type is now explicitly updated to exclude array types.

11925 Extensions beyond the ISO C standard are marked.

11926 Issue 7

11927 Austin Group Interpretation 1003.1-2001 #172 is applied, adding rationale about a conflict for the
 11928 definition of {TMP_MAX} with the ISO C standard.

11929 SD5-XBD-ERN-99 is applied, adding APPLICATION USAGE.

11930 The *dprintf()*, *fmemopen()*, *getdelim()*, *getline()*, *open_memstream()*, and *vdprintf()* functions are
 11931 added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

11932 The *renameat()* function is added from The Open Group Technical Standard, 2006, Extended API
 11933 Set Part 2.

11934 The *gets()*, *tmpnam()*, and *tempnam()* functions and the *L_tmpnam* macro are marked
 11935 obsolescent.

11936 This reference page is clarified with respect to macros and symbolic constants, and a declaration
 11937 for the **off_t** type is added.

Headers

<stdlib.h>

NAME

stdlib.h — standard library definitions

SYNOPSIS

```
#include <stdlib.h>
```

DESCRIPTION

Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to enable the visibility of these symbols in this header.

The <stdlib.h> header shall define the following macros which shall expand to integer constant expressions:

EXIT_FAILURE Unsuccessful termination for *exit()*; evaluates to a non-zero value.

EXIT_SUCCESS Successful termination for *exit()*; evaluates to 0.

{RAND_MAX} Maximum value returned by *rand()*; at least 32 767.

The <stdlib.h> header shall define the following macro which shall expand to a positive integer expression with type **size_t**:

{MB_CUR_MAX} Maximum number of bytes in a character specified by the current locale (category LC_CTYPE).

The <stdlib.h> header shall define NULL as described in <stddef.h>.

The <stdlib.h> header shall define the following data types through **typedef**:

div_t Structure type returned by the *div()* function.

ldiv_t Structure type returned by the *ldiv()* function.

lldiv_t Structure type returned by the *lldiv()* function.

size_t As described in <stddef.h>.

wchar_t As described in <stddef.h>.

In addition, the <stdlib.h> header shall define the following symbolic constants and macros as described in <sys/wait.h>:

WEXITSTATUS
WIFEXITED
WIFSIGNALED
WIFSTOPPED
WNOHANG
WSTOPSIG
WTERMSIG
WUNTRACED

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void      _Exit(int);
long      a64l(const char *);
void      abort(void);
int       abs(int);
int       atexit(void (*) (void));
double    atof(const char *);
```


<stdlib.h>

Headers

```

11980      int      atoi(const char *);
11981      long      atol(const char *);
11982      long long  atoll(const char *);
11983      void      *bsearch(const void *, const void *, size_t, size_t,
11984                        int (*)(const void *, const void *));
11985      void      *calloc(size_t, size_t);
11986      div_t      div(int, int);
11987  XSI      double drand48(void);
11988      double      erand48(unsigned short [3]);
11989      void      exit(int);
11990      void      free(void *);
11991      char      *getenv(const char *);
11992      int      getsuopt(char **, char *const *, char **);
11993  XSI      int      grantpt(int);
11994      char      *initstate(unsigned, char *, size_t);
11995      long      jrand48(unsigned short [3]);
11996      char      *l64a(long);
11997      long      labs(long);
11998  XSI      void      lcong48(unsigned short [7]);
11999      ldiv_t     ldiv(long, long);
12000      long long  llabs(long long);
12001      lldiv_t    lldiv(long long, long long);
12002  XSI      long      lrand48(void);
12003      void      *malloc(size_t);
12004      int      mblen(const char *, size_t);
12005      size_t     mbstowcs(wchar_t *restrict, const char *restrict, size_t);
12006      int      mbtowc(wchar_t *restrict, const char *restrict, size_t);
12007  CX      char      *mkdtemp(char *);
12008      int      mkstemp(char *);
12009  XSI      long      mrand48(void);
12010      long      nrand48(unsigned short [3]);
12011  ADV      int      posix_memalign(void **, size_t, size_t);
12012  XSI      int      posix_openpt(int);
12013      char      *ptsname(int);
12014      int      putenv(char *);
12015      void      qsort(void *, size_t, size_t, int (*)(const void *,
12016                        const void *));
12017      int      rand(void);
12018  OB CX     int      rand_r(unsigned *);
12019  XSI      long      random(void);
12020      void      *realloc(void *, size_t);
12021  XSI      char      *realpath(const char *restrict, char *restrict);
12022      unsigned short *seed48(unsigned short [3]);
12023  CX      int      setenv(const char *, const char *, int);
12024  XSI      void      setkey(const char *);
12025      char      *setstate(char *);
12026      void      srand(unsigned);
12027  XSI      void      srand48(long);
12028      void      srandom(unsigned);
12029      double     strtod(const char *restrict, char **restrict);
12030      float      strtodf(const char *restrict, char **restrict);
12031      long      strtol(const char *restrict, char **restrict, int);

```


Headers

<stdlib.h>

```

12032      long double      strtold(const char *restrict, char **restrict);
12033      long long        strtoll(const char *restrict, char **restrict, int);
12034      unsigned long     strtoul(const char *restrict, char **restrict, int);
12035      unsigned long long
12036                      strtoull(const char *restrict, char **restrict, int);
12037      int               system(const char *);
12038 XSI      int           unlockpt(int);
12039 CX       int           unsetenv(const char *);
12040      size_t           wcstombs(char *restrict, const wchar_t *restrict, size_t);
12041      int              wctomb(char *, wchar_t);

```

12042 CX Inclusion of the <stdlib.h> header may also make visible all symbols from <stddef.h>, <limits.h>, <math.h>, and <sys/wait.h>.

12044 APPLICATION USAGE

12045 None.

12046 RATIONALE

12047 None.

12048 FUTURE DIRECTIONS

12049 None.

12050 SEE ALSO

12051 <limits.h>, <math.h>, <stddef.h>, <sys/types.h>, <sys/wait.h>

12052 XSH Section 2.2 (on page 468), *_Exit()*, *a64l()*, *abort()*, *abs()*, *atexit()*, *atof()*, *atoi()*, *atol()*,
12053 *bsearch()*, *calloc()*, *div()*, *drand48()*, *exit()*, *free()*, *getenv()*, *getsubopt()*, *grantpt()*, *initstate()*, *labs()*,
12054 *ldiv()*, *malloc()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *mkdtemp()*, *posix_memalign()*, *posix_openpt()*,
12055 *ptsname()*, *putenv()*, *qsort()*, *rand()*, *realloc()*, *realpath()*, *setenv()*, *setkey()*, *strtod()*, *strtol()*,
12056 *strtoul()*, *system()*, *unlockpt()*, *unsetenv()*, *wcstombs()*, *wctomb()*

12057 CHANGE HISTORY

12058 First released in Issue 3.

12059 Issue 5

12060 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

12061 The *ttyslot()* and *valloc()* functions are marked LEGACY.

12062 The type of the third argument to *initstate()* is changed from **int** to **size_t**. The type of the return
12063 value from *setstate()* is changed from **char** to **char ***, and the type of the first argument is
12064 changed from **char *** to **const char ***.

12065 Issue 6

12066 The Open Group Corrigendum U021/1 is applied, correcting the prototype for *realpath()* to be
12067 consistent with the reference page.

12068 The Open Group Corrigendum U028/13 is applied, correcting the prototype for *putenv()* to be
12069 consistent with the reference page.

12070 The *rand_r()* function is marked as part of the Thread-Safe Functions option.

12071 Function prototypes for *setenv()* and *unsetenv()* are added.

12072 The *posix_memalign()* function is added for alignment with IEEE Std 1003.1d-1999.

12073 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

12074 The *ecvt()*, *fcvt()*, *gcvt()*, and *mktemp()* functions are marked LEGACY.

<stdlib.h>

Headers

- 12075 The *ttyslot()* and *valloc()* functions are removed as they were previously marked LEGACY.
- 12076 Extensions beyond the ISO C standard are marked.
- 12077 **Issue 7**
- 12078 SD5-XBD-ERN-79 and SD5-XBD-ERN-105 are applied.
- 12079 The LEGACY functions are removed.
- 12080 The *mkdtemp()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.
- 12081
- 12082 The *rand_r()* function is marked obsolescent.
- 12083 This reference page is clarified with respect to macros and symbolic constants.
- 12084 The type of the first argument to *setstate()* is changed from **const char *** to **char ***.

Headers

<string.h>

NAME

string.h — string operations

SYNOPSIS

```
#include <string.h>
```

DESCRIPTION

Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to enable the visibility of these symbols in this header.

The <string.h> header shall define NULL and **size_t** as described in <stddef.h>.

The <string.h> header shall define the **locale_t** type as described in <locale.h>.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
void *memcpy(void *restrict, const void *restrict, int, size_t);
void *memchr(const void *, int, size_t);
int memcmp(const void *, const void *, size_t);
void *memcpy(void *restrict, const void *restrict, size_t);
void *memmove(void *, const void *, size_t);
void *memset(void *, int, size_t);
char *strcpy(char *restrict, const char *restrict);
char *strncpy(char *restrict, const char *restrict, size_t);
char *strcat(char *restrict, const char *restrict);
char *strchr(const char *, int);
int strcmp(const char *, const char *);
int strcoll(const char *, const char *);
int strcoll_l(const char *, const char *, locale_t);
char *strcpy(char *restrict, const char *restrict);
size_t strcspn(const char *, const char *);
char *strdup(const char *);
char *strerror(int);
char *strerror_l(int, locale_t);
int strerror_r(int, char *, size_t);
size_t strlen(const char *);
char *strncat(char *restrict, const char *restrict, size_t);
int strncmp(const char *, const char *, size_t);
char *strncpy(char *restrict, const char *restrict, size_t);
char *strndup(const char *, size_t);
size_t strnlen(const char *, size_t);
char *strpbrk(const char *, const char *);
char *strrchr(const char *, int);
char *strsignal(int);
size_t strspn(const char *, const char *);
char *strstr(const char *, const char *);
char *strtok(char *restrict, const char *restrict);
char *strtok_r(char *restrict, const char *restrict, char **restrict);
size_t strxfrm(char *restrict, const char *restrict, size_t);
size_t strxfrm_l(char *restrict, const char *restrict,
                 size_t, locale_t);
```

<string.h>

Headers

12132 CX Inclusion of the **<string.h>** header may also make visible all symbols from **<stddef.h>**.

12133 **APPLICATION USAGE**

12134 None.

12135 **RATIONALE**

12136 None.

12137 **FUTURE DIRECTIONS**

12138 None.

12139 **SEE ALSO**

12140 **<locale.h>**, **<stddef.h>**, **<sys/types.h>**

12141 XSH Section 2.2 (on page 468), *memcpy()*, *memchr()*, *memcmp()*, *memcpy()*, *memmove()*,
 12142 *memset()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*, *strcpy()*, *strcspn()*, *strdup()*, *strerror()*, *strlen()*,
 12143 *strncat()*, *strncmp()*, *strncpy()*, *strpbrk()*, *strrchr()*, *strsignal()*, *strspn()*, *strstr()*, *strtok()*, *strxfrm()*

12144 **CHANGE HISTORY**

12145 First released in Issue 1. Derived from Issue 1 of the SVID.

12146 **Issue 5**

12147 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

12148 **Issue 6**

12149 The *strtok_r()* function is marked as part of the Thread-Safe Functions option.

12150 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

12151 The *strerror_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.

12152 **Issue 7**

12153 SD5-XBD-ERN-15 is applied, correcting the prototype for the *strerror_r()* function.

12154 The *stpncpy()*, *stpncpy()*, *strndup()*, *strlen()*, and *strsignal()* functions are added from The Open
 12155 Group Technical Standard, 2006, Extended API Set Part 1.

12156 The *strcoll_l()*, *strerror_l()*, and *strxfrm_l()* functions are added from The Open Group Technical
 12157 Standard, 2006, Extended API Set Part 4.

12158 This reference page is clarified with respect to macros and symbolic constants, and a declaration
 12159 for the **locale_t** type is added.

Headers

<strings.h>

NAME

strings.h — string operations

SYNOPSIS

```
#include <strings.h>
```

DESCRIPTION

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
XSI int ffs(int);
int strcasecmp(const char *, const char *);
int strcasecmp_l(const char *, const char *, locale_t);
int strncasecmp(const char *, const char *, size_t);
int strncasecmp_l(const char *, const char *, size_t, locale_t);
```

The <strings.h> header shall define the **locale_t** type as described in <locale.h>.

The <strings.h> header shall define the **size_t** type as described in <sys/types.h>.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<locale.h>, <sys/types.h>

XSH *ffs()*, *strcasecmp()*

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 6

The Open Group Corrigendum U021/2 is applied, correcting the prototype for *index()* to be consistent with the reference page.

The *bcmp()*, *bcopy()*, *bzero()*, *index()*, and *rindex()* functions are marked LEGACY.

Issue 7

SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size_t** type.

The LEGACY functions are removed.

The <strings.h> header is moved from the XSI option to the Base.

The *strcasecmp_l()* and *strncasecmp_l()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

A declaration for the **locale_t** type is added.

<stropts.h>

Headers

12196 **NAME**12197 stropts.h — STREAMS interface (**STREAMS**)12198 **SYNOPSIS**

12199 OB XSR #include <stropts.h>

12200 **DESCRIPTION**12201 The **<stropts.h>** header shall define the **bandinfo** structure, which shall include at least the
12202 following members:12203 int bi_flag Flushing type.
12204 unsigned char bi_pri Priority band.12205 The **<stropts.h>** header shall define the **strpeek** structure, which shall include at least the
12206 following members:12207 struct strbuf ctlbuf The control portion of the message.
12208 struct strbuf databuf The data portion of the message.
12209 t_uscalar_t flags RS_HIPRI or 0.12210 The **<stropts.h>** header shall define the **strbuf** structure, which shall include at least the
12211 following members:12212 char *buf Pointer to buffer.
12213 int len Length of data.
12214 int maxlen Maximum buffer length.12215 The **<stropts.h>** header shall define the **strfdinsert** structure, which shall include at least the
12216 following members:12217 struct strbuf ctlbuf The control portion of the message.
12218 struct strbuf databuf The data portion of the message.
12219 int fildes File descriptor of the other STREAM.
12220 t_uscalar_t flags RS_HIPRI or 0.
12221 int offset Relative location of the stored value.12222 The **<stropts.h>** header shall define the **striocctl** structure, which shall include at least the
12223 following members:12224 int ic_cmd *ioctl()* command.
12225 char *ic_dp Pointer to buffer.
12226 int ic_len Length of data.
12227 int ic_timeout Timeout for response.12228 The **<stropts.h>** header shall define the **strrecvfd** structure, which shall include at least the
12229 following members:12230 int fd Received file descriptor.
12231 gid_t gid GID of sender.
12232 uid_t uid UID of sender.12233 The **<stropts.h>** header shall define the **uid_t** and **gid_t** types through **typedef**, as described in
12234 **<sys/types.h>**.12235 The **<stropts.h>** header shall define the **t_scalar_t** and **t_uscalar_t** types, respectively, as signed
12236 and unsigned opaque types of equal length of at least 32 bits.12237 The **<stropts.h>** header shall define the **str_list** structure, which shall include at least the
12238 following members:

Headers

<stropts.h>

12239	struct str_mlist	*sl_modlist	STREAMS module names.
12240	int	sl_nmods	Number of STREAMS module names.
12241	The <stropts.h> header shall define the str_mlist structure, which shall include at least the		
12242	following member:		
12243	char	l_name[FMNAMESZ+1]	A STREAMS module name.
12244	The <stropts.h> header shall define at least the following symbolic constants for use as the		
12245	request argument to <i>ioctl()</i> :		
12246	I_ATMARK	Is the top message “marked”?	
12247	I_CANPUT	Is a band writable?	
12248	I_CKBAND	See if any messages exist in a band.	
12249	I_FDINSERT	Send implementation-defined information about another STREAM.	
12250	I_FIND	Look for a STREAMS module.	
12251	I_FLUSH	Flush a STREAM.	
12252	I_FLUSHBAND	Flush one band of a STREAM.	
12253	I_GETBAND	Get the band of the top message on a STREAM.	
12254	I_GETCLTIME	Get close time delay.	
12255	I_GETSIG	Retrieve current notification signals.	
12256	I_GRDOPT	Get the read mode.	
12257	I_GWROPT	Get the write mode.	
12258	I_LINK	Connect two STREAMs.	
12259	I_LIST	Get all the module names on a STREAM.	
12260	I_LOOK	Get the top module name.	
12261	I_NREAD	Size the top message.	
12262	I_PEEK	Peek at the top message on a STREAM.	
12263	I_PLINK	Persistently connect two STREAMs.	
12264	I_POP	Pop a STREAMS module.	
12265	I_PUNLINK	Dismantle a persistent STREAMS link.	
12266	I_PUSH	Push a STREAMS module.	
12267	I_RECVFD	Get a file descriptor sent via I_SENDFD.	
12268	I_SENDFD	Pass a file descriptor through a STREAMS pipe.	
12269	I_SETCLTIME	Set close time delay.	
12270	I_SETSIG	Ask for notification signals.	
12271	I_SRDOPT	Set the read mode.	
12272	I_STR	Send a STREAMS <i>ioctl()</i> .	

<stropts.h>

Headers

12273	I_SWROPT	Set the write mode.
12274	I_UNLINK	Disconnect two STREAMs.
12275	The <stropts.h> header shall define at least the following symbolic constant for use with	
12276	I_LOOK:	
12277	FMNAMESZ	The minimum size in bytes of the buffer referred to by the <i>arg</i> argument.
12278	The <stropts.h> header shall define at least the following symbolic constants for use with	
12279	I_FLUSH:	
12280	FLUSHR	Flush read queues.
12281	FLUSHRW	Flush read and write queues.
12282	FLUSHW	Flush write queues.
12283	The <stropts.h> header shall define at least the following symbolic constants for use with	
12284	I_SETSIG:	
12285	S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.
12286		
12287		
12288	S_ERROR	Notification of an error condition reaches the STREAM head.
12289	S_HANGUP	Notification of a hangup reaches the STREAM head.
12290	S_HIPRI	A high-priority message is present on a STREAM head read queue.
12291	S_INPUT	A message, other than a high-priority message, has arrived at the head of a STREAM head read queue.
12292		
12293	S_MSG	A STREAMS signal message that contains the SIGPOLL signal reaches the front of the STREAM head read queue.
12294		
12295	S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.
12296		
12297		
12298	S_RDBAND	A message with a non-zero priority band has arrived at the head of a STREAM head read queue.
12299		
12300	S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a STREAM head read queue.
12301		
12302	S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is no longer full.
12303		
12304	S_WRNORM	Equivalent to S_OUTPUT.
12305	The <stropts.h> header shall define at least the following symbolic constant for use with	
12306	I_PEEK:	
12307	RS_HIPRI	Only look for high-priority messages.
12308	The <stropts.h> header shall define at least the following symbolic constants for use with	
12309	I_SRDOPT:	
12310	RMSGD	Message-discard mode.

Headers

<stropts.h>

12311	RMSGN	Message-non-discard mode.
12312	RNORM	Byte-STREAM mode, the default.
12313	RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .
12314	RPROTDIS	Discard the control part of a message, delivering any data part, when a process issues a <i>read()</i> .
12315		
12316	RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.
12317		
12318	The <stropts.h> header shall define at least the following symbolic constant for use with	
12319	L_SWOPT:	
12320	SNDZERO	Send a zero-length message downstream when a <i>write()</i> of 0 bytes occurs.
12321	The <stropts.h> header shall define at least the following symbolic constants for use with	
12322	I_ATMARK:	
12323	ANYMARK	Check if the message is marked.
12324	LASTMARK	Check if the message is the last one marked on the queue.
12325	The <stropts.h> header shall define at least the following symbolic constant for use with	
12326	I_UNLINK:	
12327	MUXID_ALL	Unlink all STREAMs linked to the STREAM associated with <i>fildev</i> .
12328	The <stropts.h> header shall define the following symbolic constants for <i>getmsg()</i> , <i>getpmsg()</i> ,	
12329	<i>putmsg()</i> , and <i>putpmsg()</i> :	
12330	MORECTL	More control information is left in message.
12331	MOREDATA	More data is left in message.
12332	MSG_ANY	Receive any message.
12333	MSG_BAND	Receive message from specified band.
12334	MSG_HIPRI	Send/receive high-priority message.
12335	The <stropts.h> header may make visible all of the symbols from <unistd.h>.	
12336	The following shall be declared as functions and may also be defined as macros. Function	
12337	prototypes shall be provided.	
12338	<code>int fattach(int, const char *);</code>	
12339	<code>int fdetach(const char *);</code>	
12340	<code>int getmsg(int, struct strbuf *restrict, struct strbuf *restrict,</code>	
12341	<code>int *restrict);</code>	
12342	<code>int getpmsg(int, struct strbuf *restrict, struct strbuf *restrict,</code>	
12343	<code>int *restrict, int *restrict);</code>	
12344	<code>int ioctl(int, int, ...);</code>	
12345	<code>int isastream(int);</code>	
12346	<code>int putmsg(int, const struct strbuf *, const struct strbuf *, int);</code>	
12347	<code>int putpmsg(int, const struct strbuf *, const struct strbuf *, int,</code>	
12348	<code>int);</code>	

<stropts.h>

Headers

12349 **APPLICATION USAGE**

12350 None.

12351 **RATIONALE**

12352 None.

12353 **FUTURE DIRECTIONS**

12354 None.

12355 **SEE ALSO**12356 **<sys/types.h>**, **<unistd.h>**

12357 XSH *close()*, *fattach()*, *fcntl()*, *fdetach()*, *getmsg()*, *ioctl()*, *isastream()*, *open()*, *pipe()*, *read()*, *poll()*,
 12358 *putmsg()*, *signal()*, *write()*

12359 **CHANGE HISTORY**

12360 First released in Issue 4, Version 2.

12361 **Issue 5**

12362 The *flags* members of the **strpeek** and **strfdinsert** structures are changed from **type long** to
 12363 **t_uscalar_t**.

12364 **Issue 6**

12365 This header is marked as part of the XSI STREAMS Option Group.

12366 The **restrict** keyword is added to the prototypes for *getmsg()* and *getpmsg()*.12367 **Issue 7**12368 SD5-XBD-ERN-87 is applied, correcting an error in the **strecvfd** structure.12369 The **<stropts.h>** header is marked obsolescent.

12370 This reference page is clarified with respect to macros and symbolic constants.

Headers

<sys/ipc.h>

NAME

sys/ipc.h — XSI interprocess communication access structure

SYNOPSIS

```
XSI #include <sys/ipc.h>
```

DESCRIPTION

The <sys/ipc.h> header is used by three mechanisms for XSI interprocess communication (IPC): messages, semaphores, and shared memory. All use a common structure type, **ipc_perm**, to pass information used in determining permission to perform an IPC operation.

The <sys/ipc.h> header shall define the **ipc_perm** structure, which shall include the following members:

uid_t	uid	Owner's user ID.
gid_t	gid	Owner's group ID.
uid_t	cuid	Creator's user ID.
gid_t	cgid	Creator's group ID.
mode_t	mode	Read/write permission.

The <sys/ipc.h> header shall define the **uid_t**, **gid_t**, **mode_t**, and **key_t** types as described in <sys/types.h>.

The <sys/ipc.h> header shall define the following symbolic constants.

Mode bits:

IPC_CREAT	Create entry if key does not exist.
IPC_EXCL	Fail if key exists.
IPC_NOWAIT	Error if request must wait.

Keys:

IPC_PRIVATE	Private key.

Control commands:

IPC_RMID	Remove identifier.
IPC_SET	Set options.
IPC_STAT	Get options.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
key_t ftok(const char *, int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

<sys/ipc.h>

Headers

12408	SEE ALSO
12409	<sys/types.h>
12410	XSH <i>ftok</i> ()
12411	CHANGE HISTORY
12412	First released in Issue 2. Derived from System V Release 2.0.
12413	Issue 7
12414	This reference page is clarified with respect to macros and symbolic constants.

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

Headers

<sys/mman.h>

12415 NAME

12416 sys/mman.h — memory management declarations

12417 SYNOPSIS

12418 #include <sys/mman.h>

12419 DESCRIPTION

12420 The <sys/mman.h> header shall define the following symbolic constants for use as protection
12421 options:

12422 PROT_EXEC Page can be executed.

12423 PROT_NONE Page cannot be accessed.

12424 PROT_READ Page can be read.

12425 PROT_WRITE Page can be written.

12426 The <sys/mman.h> header shall define the following symbolic constants for use as flag options:

12427 MAP_FIXED Interpret *addr* exactly.

12428 MAP_PRIVATE Changes are private.

12429 MAP_SHARED Share changes.

12430 XSI|SIO The <sys/mman.h> header shall define the following symbolic constants for the *msync()*
12431 function:

12432 MS_ASYNC Perform asynchronous writes.

12433 MS_INVALIDATE Invalidate mappings.

12434 MS_SYNC Perform synchronous writes.

12435 ML The <sys/mman.h> header shall define the following symbolic constants for the *mlockall()*
12436 function:

12437 MCL_CURRENT Lock currently mapped pages.

12438 MCL_FUTURE Lock pages that become mapped.

12439 The <sys/mman.h> header shall define the symbolic constant MAP_FAILED which shall have
12440 type **void *** and shall be used to indicate a failure from the *mmap()* function .

12441 ADV If the Advisory Information option is supported, the <sys/mman.h> header shall define
12442 symbolic constants for the *advice* argument to the *posix_madvise()* function as follows:

12443 POSIX_MADV_DONTNEED

12444 The application expects that it will not access the specified range in the near future.

12445 POSIX_MADV_NORMAL

12446 The application has no advice to give on its behavior with respect to the specified range. It
12447 is the default characteristic if no advice is given for a range of memory.

12448 POSIX_MADV_RANDOM

12449 The application expects to access the specified range in a random order.

12450 POSIX_MADV_SEQUENTIAL

12451 The application expects to access the specified range sequentially from lower addresses to
12452 higher addresses.

<sys/mman.h>

Headers

12453		POSIX_MADV_WILLNEED
12454		The application expects to access the specified range in the near future.
12455	TYM	The <sys/mman.h> header shall define the following symbolic constants for use as flags for the <i>posix_typed_mem_open()</i> function:
12456		
12457		POSIX_TYPED_MEM_ALLOCATE
12458		Allocate on <i>mmap()</i> .
12459		POSIX_TYPED_MEM_ALLOCATE_CONTIG
12460		Allocate contiguously on <i>mmap()</i> .
12461		POSIX_TYPED_MEM_MAP_ALLOCATABLE
12462		Map on <i>mmap()</i> , without affecting allocatability.
12463		The <sys/mman.h> header shall define the mode_t , off_t , and size_t types as described in <sys/types.h>.
12464		
12465	TYM	The <sys/mman.h> header shall define the posix_typed_mem_info structure, which shall include at least the following member:
12466		
12467		size_t posix_tmi_length Maximum length which may be allocated from a typed memory object.
12468		
12469		The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.
12470		
12471	MLR	int mlock(const void *, size_t);
12472	ML	int mlockall(int);
12473		void *mmap(void *, size_t, int, int, int, off_t);
12474		int mprotect(void *, size_t, int);
12475	XSI SIO	int msync(void *, size_t, int);
12476	MLR	int munlock(const void *, size_t);
12477	ML	int munlockall(void);
12478		int munmap(void *, size_t);
12479	ADV	int posix_madvise(void *, size_t, int);
12480	TYM	int posix_mem_offset(const void *restrict, size_t, off_t *restrict, size_t *restrict, int *restrict);
12481		
12482		int posix_typed_mem_get_info(int, struct posix_typed_mem_info *);
12483		int posix_typed_mem_open(const char *, int, int);
12484	SHM	int shm_open(const char *, int, mode_t);
12485		int shm_unlink(const char *);

Headers

<sys/mman.h>

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *mlock()*, *mlockall()*, *mmap()*, *mprotect()*, *msync()*, *munmap()*, *posix_madvise()*,
posix_mem_offset(), *posix_typed_mem_get_info()*, *posix_typed_mem_open()*, *shm_open()*,
shm_unlink()

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Updated for alignment with the POSIX Realtime Extension.

Issue 6

The <sys/mman.h> header is marked as dependent on support for either the Memory Mapped Files, Process Memory Locking, or Shared Memory Objects options.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The TYM margin code is added to the list of margin codes for the <sys/mman.h> header line, as well as for other lines.
- The POSIX_TYPED_MEM_ALLOCATE, POSIX_TYPED_MEM_ALLOCATE_CONTIG, and POSIX_TYPED_MEM_MAP_ALLOCATABLE flags are added.
- The **posix_tmi_length** structure is added.
- The *posix_mem_offset()*, *posix_typed_mem_get_info()*, and *posix_typed_mem_open()* functions are added.

The **restrict** keyword is added to the prototype for *posix_mem_offset()*.

IEEE PASC Interpretation 1003.1 #102 is applied, adding the prototype for *posix_madvise()*.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/16 is applied, correcting margin code and shading errors for the *mlock()* and *munlock()* functions.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code for the *mmap()* function from MF|SHM to MC3 (notation for MF|SHM|TYM).

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code for the *munmap()* function from MF|SHM to MC3 (notation for MF|SHM|TYM).

Issue 7

SD5-XBD-ERN-5 is applied, rewriting the DESCRIPTION.

Functionality relating to the Memory Protection and Memory Mapped Files options is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants.

<sys/msg.h>

Headers

12525 NAME

12526 sys/msg.h — XSI message queue structures

12527 SYNOPSIS

12528 XSI `#include <sys/msg.h>`

12529 DESCRIPTION

12530 The <sys/msg.h> header shall define the following data types through **typedef**:12531 **msgqnum_t** Used for the number of messages in the message queue.12532 **msglen_t** Used for the number of bytes allowed in a message queue.12533 These types shall be unsigned integer types that are able to store values at least as large as a type
12534 **unsigned short**.12535 The <sys/msg.h> header shall define the following symbolic constant as a message operation
12536 flag:12537 **MSG_NOERROR** No error if big message.12538 The <sys/msg.h> header shall define the **msqid_ds** structure, which shall include the following
12539 members:

12540	<code>struct ipc_perm</code>	<code>msg_perm</code>	Operation permission structure.
12541	<code>msgqnum_t</code>	<code>msg_qnum</code>	Number of messages currently on queue.
12542	<code>msglen_t</code>	<code>msg_qbytes</code>	Maximum number of bytes allowed on queue.
12543	<code>pid_t</code>	<code>msg_lspid</code>	Process ID of last <i>msgsnd()</i> .
12544	<code>pid_t</code>	<code>msg_lrpid</code>	Process ID of last <i>msgrcv()</i> .
12545	<code>time_t</code>	<code>msg_stime</code>	Time of last <i>msgsnd()</i> .
12546	<code>time_t</code>	<code>msg_rtime</code>	Time of last <i>msgrcv()</i> .
12547	<code>time_t</code>	<code>msg_ctime</code>	Time of last change.

12548 The <sys/msg.h> header shall define the **pid_t**, **size_t**, **ssize_t**, and **time_t** types as described in
12549 <sys/types.h>.12550 The following shall be declared as functions and may also be defined as macros. Function
12551 prototypes shall be provided.

```

12552 int      msgctl(int, int, struct msqid_ds *);
12553 int      msgget(key_t, int);
12554 ssize_t  msgrcv(int, void *, size_t, long, int);
12555 int      msgsnd(int, const void *, size_t, int);

```

12556 In addition, the <sys/msg.h> header shall include the <sys/ipc.h> header.

12557 APPLICATION USAGE

12558 None.

12559 RATIONALE

12560 None.

12561 FUTURE DIRECTIONS

12562 None.

12563 SEE ALSO

12564 <sys/ipc.h>, <sys/types.h>

12565 XSH *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

Headers

<sys/msg.h>

12566 **CHANGE HISTORY**

12567 First released in Issue 2. Derived from System V Release 2.0.

12568 **Issue 7**

12569 Austin Group Interpretation 1003.1-2001 #179 is applied.

12570 This reference page is clarified with respect to macros and symbolic constants.

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

<sys/resource.h>

Headers

12571 NAME

12572 sys/resource.h — definitions for XSI resource operations

12573 SYNOPSIS

12574 XSI `#include <sys/resource.h>`

12575 DESCRIPTION

12576 The <sys/resource.h> header shall define the following symbolic constants as possible values of
12577 the *which* argument of *getpriority()* and *setpriority()*:

12578 PRIO_PROCESS Identifies the *who* argument as a process ID.

12579 PRIO_PGRP Identifies the *who* argument as a process group ID.

12580 PRIO_USER Identifies the *who* argument as a user ID.

12581 The <sys/resource.h> header shall define the following type through **typedef**:

12582 **rlim_t** Unsigned integer type used for limit values.

12583 The <sys/resource.h> header shall define the following symbolic constants, which shall have
12584 values suitable for use in **#if** preprocessing directives:

12585 RLIM_INFINITY A value of **rlim_t** indicating no limit.

12586 RLIM_SAVED_MAX A value of type **rlim_t** indicating an unrepresentable saved hard
12587 limit.

12588 RLIM_SAVED_CUR A value of type **rlim_t** indicating an unrepresentable saved soft limit.

12589 On implementations where all resource limits are representable in an object of type **rlim_t**,
12590 RLIM_SAVED_MAX and RLIM_SAVED_CUR need not be distinct from RLIM_INFINITY.

12591 The <sys/resource.h> header shall define the following symbolic constants as possible values of
12592 the *who* parameter of *getrusage()*:

12593 RUSAGE_SELF Returns information about the current process.

12594 RUSAGE_CHILDREN Returns information about children of the current process.

12595 The <sys/resource.h> header shall define the **rlimit** structure, which shall include at least the
12596 following members:

12597 `rlim_t rlim_cur` The current (soft) limit.

12598 `rlim_t rlim_max` The hard limit.

12599 The <sys/resource.h> header shall define the **rusage** structure, which shall include at least the
12600 following members:

12601 `struct timeval ru_utime` User time used.

12602 `struct timeval ru_stime` System time used.

12603 The <sys/resource.h> header shall define the **timeval** structure as described in <sys/time.h>.

12604 The <sys/resource.h> header shall define the following symbolic constants as possible values for
12605 the *resource* argument of *getrlimit()* and *setrlimit()*:

12606 RLIMIT_CORE Limit on size of **core** file.

12607 RLIMIT_CPU Limit on CPU time per process.

Headers

<sys/resource.h>

12608	RLIMIT_DATA	Limit on data segment size.
12609	RLIMIT_FSIZE	Limit on file size.
12610	RLIMIT_NOFILE	Limit on number of open files.
12611	RLIMIT_STACK	Limit on stack size.
12612	RLIMIT_AS	Limit on address space size.
12613	The following shall be declared as functions and may also be defined as macros. Function	
12614	prototypes shall be provided.	
12615	<code>int getpriority(int, id_t);</code>	
12616	<code>int getrlimit(int, struct rlimit *);</code>	
12617	<code>int getrusage(int, struct rusage *);</code>	
12618	<code>int setpriority(int, id_t, int);</code>	
12619	<code>int setrlimit(int, const struct rlimit *);</code>	
12620	The <sys/resource.h> header shall define the <code>id_t</code> type through <code>typedef</code> , as described in	
12621	<sys/types.h>.	
12622	Inclusion of the <sys/resource.h> header may also make visible all symbols from <sys/time.h>.	
12623	APPLICATION USAGE	
12624	None.	
12625	RATIONALE	
12626	None.	
12627	FUTURE DIRECTIONS	
12628	None.	
12629	SEE ALSO	
12630	<sys/time.h>, <sys/types.h>	
12631	XSH <code>getpriority()</code> , <code>getrlimit()</code> , <code>getrusage()</code>	
12632	CHANGE HISTORY	
12633	First released in Issue 4, Version 2.	
12634	Issue 5	
12635	Large File System extensions are added.	
12636	Issue 7	
12637	This reference page is clarified with respect to macros and symbolic constants.	

<sys/select.h>

Headers

12638 **NAME**

12639 sys/select.h — select types

12640 **SYNOPSIS**

12641 #include <sys/select.h>

12642 **DESCRIPTION**

12643 The **<sys/select.h>** header shall define the **timeval** structure, which shall include at least the
 12644 following members:

12645	time_t	tv_sec	Seconds.
12646	suseconds_t	tv_usec	Microseconds.

12647 The **<sys/select.h>** header shall define the **time_t** and **suseconds_t** types as described in
 12648 **<sys/types.h>**.

12649 The **<sys/select.h>** header shall define the **sigset_t** type as described in **<signal.h>**.

12650 The **<sys/select.h>** header shall define the **timespec** structure as described in **<time.h>**.

12651 The **<sys/select.h>** header shall define the **fd_set** type as a structure.

12652 The **<sys/select.h>** header shall define the following symbolic constant, which shall have a value
 12653 suitable for use in **#if** preprocessing directives:

12654 **FD_SETSIZE** Maximum number of file descriptors in an **fd_set** structure.

12655 The following shall be declared as functions, defined as macros, or both. If functions are
 12656 declared, function prototypes shall be provided.

```
12657 void FD_CLR(int, fd_set *);
12658 int FD_ISSET(int, fd_set *);
12659 void FD_SET(int, fd_set *);
12660 void FD_ZERO(fd_set *);
```

12661 If implemented as macros, these may evaluate their arguments more than once, so applications
 12662 should ensure that the arguments they supply are never expressions with side-effects.

12663 The following shall be declared as functions and may also be defined as macros. Function
 12664 prototypes shall be provided.

```
12665 int pselect(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12666             const struct timespec *restrict, const sigset_t *restrict);
12667 int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12668            struct timeval *restrict);
```

12669 Inclusion of the **<sys/select.h>** header may make visible all symbols from the headers
 12670 **<signal.h>** and **<time.h>**.

12671 **APPLICATION USAGE**

12672 None.

12673 **RATIONALE**

12674 None.

12675 **FUTURE DIRECTIONS**

12676 None.

Headers

<sys/select.h>

12677 **SEE ALSO**

12678 <signal.h>, <sys/time.h>, <sys/types.h>, <time.h>

12679 XSH *pselect()*12680 **CHANGE HISTORY**

12681 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

12682 The requirement for the **fd_set** structure to have a member *fds_bits* has been removed as per The
12683 Open Group Base Resolution bwg2001-005.12684 **Issue 7**

12685 SD5-XBD-ERN-6 is applied, reordering the DESCRIPTION.

12686 This reference page is clarified with respect to macros and symbolic constants.

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

<sys/sem.h>

Headers

12687 NAME

12688 sys/sem.h — XSI semaphore facility

12689 SYNOPSIS

12690 XSI `#include <sys/sem.h>`

12691 DESCRIPTION

12692 The <sys/sem.h> header shall define the following symbolic constant for use as a semaphore
12693 operation flag:

12694 SEM_UNDO Set up adjust on exit entry.

12695 The <sys/sem.h> header shall define the following symbolic constants for use as commands for
12696 the *semctl()* function:12697 GETNCNT Get *semncnt*.12698 GETPID Get *sempid*.12699 GETVAL Get *semval*.12700 GETALL Get all cases of *semval*.12701 GETZCNT Get *semzcnt*.12702 SETVAL Set *semval*.12703 SETALL Set all cases of *semval*.12704 The <sys/sem.h> header shall define the **semid_ds** structure, which shall include the following
12705 members:

12706 struct ipc_perm sem_perm Operation permission structure.

12707 unsigned short sem_nsems Number of semaphores in set.

12708 time_t sem_otime Last *semop()* time.12709 time_t sem_ctime Last time changed by *semctl()*.12710 The <sys/sem.h> header shall define the **pid_t**, **size_t**, and **time_t** types as described in
12711 <sys/types.h>.12712 A semaphore shall be represented by an anonymous structure, which shall include the following
12713 members:

12714 unsigned short semval Semaphore value.

12715 pid_t sempid Process ID of last operation.

12716 unsigned short semncnt Number of processes waiting for *semval*
12717 to become greater than current value.12718 unsigned short semzcnt Number of processes waiting for *semval*
12719 to become 0.12720 The <sys/sem.h> header shall define the **sembuf** structure, which shall include the following
12721 members:

12722 unsigned short sem_num Semaphore number.

12723 short sem_op Semaphore operation.

12724 short sem_flg Operation flags.

12725 The following shall be declared as functions and may also be defined as macros. Function
12726 prototypes shall be provided.

12727 int semctl(int, int, int, ...);

*Headers***<sys/sem.h>**

```

12728         int    semget(key_t, int, int);
12729         int    semop(int, struct sembuf *, size_t);

```

12730 In addition, the **<sys/sem.h>** header shall include the **<sys/ipc.h>** header.

12731 **APPLICATION USAGE**

12732 None.

12733 **RATIONALE**

12734 None.

12735 **FUTURE DIRECTIONS**

12736 None.

12737 **SEE ALSO**

12738 **<sys/ipc.h>**, **<sys/types.h>**

12739 XSH *semctl()*, *semget()*, *semop()*

12740 **CHANGE HISTORY**

12741 First released in Issue 2. Derived from System V Release 2.0.

12742 **Issue 7**

12743 Austin Group Interpretation 1003.1-2001 #179 is applied.

12744 This reference page is clarified with respect to macros and symbolic constants.

<sys/shm.h>

Headers

12745 NAME

12746 sys/shm.h — XSI shared memory facility

12747 SYNOPSIS

12748 XSI `#include <sys/shm.h>`

12749 DESCRIPTION

12750 The <sys/shm.h> header shall define the following symbolic constants:

12751 SHM_RDONLY Attach read-only (else read-write).

12752 SHM_RND Round attach address to SHMLBA.

12753 SHMLBA Segment low boundary address multiple.

12754 The <sys/shm.h> header shall define the following data types through **typedef**:12755 **shmatt_t** Unsigned integer used for the number of current attaches that must be able to
12756 store values at least as large as a type **unsigned short**.12757 The <sys/shm.h> header shall define the **shmid_ds** structure, which shall include the following
12758 members:

12759	<code>struct ipc_perm</code>	<code>shm_perm</code>	Operation permission structure.
12760	<code>size_t</code>	<code>shm_segsz</code>	Size of segment in bytes.
12761	<code>pid_t</code>	<code>shm_lpid</code>	Process ID of last shared memory operation.
12762	<code>pid_t</code>	<code>shm_cpid</code>	Process ID of creator.
12763	<code>shmatt_t</code>	<code>shm_nattch</code>	Number of current attaches.
12764	<code>time_t</code>	<code>shm_atime</code>	Time of last <code>shmat()</code> .
12765	<code>time_t</code>	<code>shm_dtime</code>	Time of last <code>shmdt()</code> .
12766	<code>time_t</code>	<code>shm_ctime</code>	Time of last change by <code>shmctl()</code> .

12767 The <sys/shm.h> header shall define the **pid_t**, **size_t**, and **time_t** types as described in
12768 <sys/types.h>.12769 The following shall be declared as functions and may also be defined as macros. Function
12770 prototypes shall be provided:

```

12771 void *shmat(int, const void *, int);
12772 int shmctl(int, int, struct shmid_ds *);
12773 int shmdt(const void *);
12774 int shmget(key_t, size_t, int);

```

12775 In addition, the <sys/shm.h> header shall include the <sys/ipc.h> header.

12776 APPLICATION USAGE

12777 None.

12778 RATIONALE

12779 None.

12780 FUTURE DIRECTIONS

12781 None.

12782 SEE ALSO

12783 <sys/ipc.h>, <sys/types.h>

12784 XSH `shmat()`, `shmctl()`, `shmdt()`, `shmget()`

12785 **CHANGE HISTORY**

12786 First released in Issue 2. Derived from System V Release 2.0.

12787 **Issue 5**

12788 The type of *shm_segsz* is changed from **int** to **size_t**.

12789 **Issue 7**

12790 Austin Group Interpretation 1003.1-2001 #179 is applied.

12791 This reference page is clarified with respect to macros and symbolic constants.

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

<sys/socket.h>

Headers

12792 NAME

12793 sys/socket.h — main sockets header

12794 SYNOPSIS

12795 #include <sys/socket.h>

12796 DESCRIPTION

12797 The <sys/socket.h> header shall define the **socklen_t** type, which is an integer type of width of
12798 at least 32 bits; see APPLICATION USAGE.12799 The <sys/socket.h> header shall define the **sa_family_t** unsigned integer type.12800 The <sys/socket.h> header shall define the **sockaddr** structure, which shall include at least the
12801 following members:

12802	sa_family_t	sa_family	Address family.
12803	char	sa_data[]	Socket address (variable-length data).

12804 The **sockaddr** structure is used to define a socket address which is used in the *bind()*, *connect()*,
12805 *getpeername()*, *getsockname()*, *recvfrom()*, and *sendto()* functions.12806 The <sys/socket.h> header shall define the **sockaddr_storage** structure, which shall be:

- 12807 • Large enough to accommodate all supported protocol-specific address structures
- 12808 • Aligned at an appropriate boundary so that pointers to it can be cast as pointers to
12809 protocol-specific address structures and used to access the fields of those structures
12810 without alignment problems

12811 The **sockaddr_storage** structure shall include at least the following members:12812 **sa_family_t** **ss_family**

12813 When a **sockaddr_storage** structure is cast as a **sockaddr** structure, the *ss_family* field of the
12814 **sockaddr_storage** structure shall map onto the *sa_family* field of the **sockaddr** structure. When a
12815 **sockaddr_storage** structure is cast as a protocol-specific address structure, the *ss_family* field
12816 shall map onto a field of that structure that is of type **sa_family_t** and that identifies the
12817 protocol's address family.

12818 The <sys/socket.h> header shall define the **msghdr** structure, which shall include at least the
12819 following members:

12820	void	*msg_name	Optional address.
12821	socklen_t	msg_namelen	Size of address.
12822	struct iovec	*msg_iov	Scatter/gather array.
12823	int	msg_iovlen	Members in <i>msg_iov</i> .
12824	void	*msg_control	Ancillary data; see below.
12825	socklen_t	msg_controllen	Ancillary data buffer <i>len</i> .
12826	int	msg_flags	Flags on received message.

12827 The **msghdr** structure is used to minimize the number of directly supplied parameters to the
12828 *recvmsg()* and *sendmsg()* functions. This structure is used as a *value-result* parameter in the
12829 *recvmsg()* function and *value* only for the *sendmsg()* function.

12830 The <sys/socket.h> header shall define the **iovec** structure as described in <sys/uio.h>.12831 The <sys/socket.h> header shall define the **cmsg_hdr** structure, which shall include at least the
12832 following members:

12833	socklen_t	cmsg_len	Data byte count, including the cmsg_hdr .
12834	int	cmsg_level	Originating protocol.

Headers

<sys/socket.h>

12835 int cmsg_type Protocol-specific type.

12836 The cmsghdr structure is used for storage of ancillary data object information.

12837 Ancillary data consists of a sequence of pairs, each consisting of a cmsghdr structure followed

12838 by a data array. The data array contains the ancillary data message, and the cmsghdr structure

12839 contains descriptive information that allows an application to correctly parse the data.

12840 The values for cmsg_level shall be legal values for the level argument to the getsockopt() and

12841 setsockopt() functions. The system documentation shall specify the cmsg_type definitions for the

12842 supported protocols.

12843 Ancillary data is also possible at the socket level. The <sys/socket.h> header shall define the

12844 following symbolic constant for use as the cmsg_type value when cmsg_level is SOL_SOCKET:

12845 SCM_RIGHTS Indicates that the data array contains the access rights to be sent or

12846 received.

12847 The <sys/socket.h> header shall define the following macros to gain access to the data arrays in

12848 the ancillary data associated with a message header:

12849 CMSG_DATA(cmsg)

12850 If the argument is a pointer to a cmsghdr structure, this macro shall return an unsigned

12851 character pointer to the data array associated with the cmsghdr structure.

12852 CMSG_NXTHDR(mhdr,cmsg)

12853 If the first argument is a pointer to a msghdr structure and the second argument is a pointer

12854 to a cmsghdr structure in the ancillary data pointed to by the msg_control field of that

12855 msghdr structure, this macro shall return a pointer to the next cmsghdr structure, or a null

12856 pointer if this structure is the last cmsghdr in the ancillary data.

12857 CMSG_FIRSTHDR(mhdr)

12858 If the argument is a pointer to a msghdr structure, this macro shall return a pointer to the

12859 first cmsghdr structure in the ancillary data associated with this msghdr structure, or a null

12860 pointer if there is no ancillary data associated with the msghdr structure.

12861 The <sys/socket.h> header shall define the linger structure, which shall include at least the

12862 following members:

12863 int l_onoff Indicates whether linger option is enabled.

12864 int l_linger Linger time, in seconds.

12865 The <sys/socket.h> header shall define the following symbolic constants with distinct values:

12866 SOCK_DGRAM Datagram socket.

12867 RS SOCK_RAW Raw Protocol Interface.

12868 SOCK_SEQPACKET Sequenced-packet socket.

12869 SOCK_STREAM Byte-stream socket.

12870 The <sys/socket.h> header shall define the following symbolic constant for use as the level

12871 argument of setsockopt() and getsockopt().

12872 SOL_SOCKET Options to be accessed at socket level, not protocol level.

12873 The <sys/socket.h> header shall define the following symbolic constants with distinct values for

12874 use as the option_name argument in getsockopt() or setsockopt() calls (see XSH Section 2.10.16, on

12875 page 522):

<sys/socket.h>

Headers

12876	SO_ACCEPTCONN	Socket is accepting connections.
12877	SO_BROADCAST	Transmission of broadcast messages is supported.
12878	SO_DEBUG	Debugging information is being recorded.
12879	SO_DONTROUTE	Bypass normal routing.
12880	SO_ERROR	Socket error status.
12881	SO_KEEPALIVE	Connections are kept alive with periodic messages.
12882	SO_LINGER	Socket lingers on close.
12883	SO_OOBINLINE	Out-of-band data is transmitted in line.
12884	SO_RCVBUF	Receive buffer size.
12885	SO_RCVLOWAT	Receive “low water mark”.
12886	SO_RCVTIMEO	Receive timeout.
12887	SO_REUSEADDR	Reuse of local addresses is supported.
12888	SO_SNDBUF	Send buffer size.
12889	SO_SNDLOWAT	Send “low water mark”.
12890	SO_SNDTIMEO	Send timeout.
12891	SO_TYPE	Socket type.
12892	The <sys/socket.h> header shall define the following symbolic constant for use as the maximum	
12893	<i>backlog</i> queue length which may be specified by the <i>backlog</i> field of the <i>listen()</i> function:	
12894	SOMAXCONN	The maximum <i>backlog</i> queue length.
12895	The <sys/socket.h> header shall define the following symbolic constants with distinct values for	
12896	use as the valid values for the <i>msg_flags</i> field in the msghdr structure, or the <i>flags</i> parameter in	
12897	<i>recv()</i> , <i>recvfrom()</i> , <i>recvmsg()</i> , <i>send()</i> , <i>sendmsg()</i> , or <i>sendto()</i> calls:	
12898	MSG_CTRUNC	Control data truncated.
12899	MSG_DONTROUTE	Send without using routing tables.
12900	MSG_EOR	Terminates a record (if supported by the protocol).
12901	MSG_OOB	Out-of-band data.
12902	MSG_NOSIGNAL	No SIGPIPE generated when an attempt to send is made on a stream-oriented socket that is no longer connected.
12903		
12904	MSG_PEEK	Leave received data in queue.
12905	MSG_TRUNC	Normal data truncated.
12906	MSG_WAITALL	Attempt to fill the read buffer.
12907	The <sys/socket.h> header shall define the following symbolic constants with distinct values:	
12908	AF_INET	Internet domain sockets for use with IPv4 addresses.
12909	IP6	AF_INET6 Internet domain sockets for use with IPv6 addresses.
12910	AF_UNIX	UNIX domain sockets.

Headers

<sys/socket.h>

12911 AF_UNSPEC Unspecified.

12912 The <sys/socket.h> header shall define the following symbolic constants with distinct values:

12913 SHUT_RD Disables further receive operations.

12914 SHUT_RDWR Disables further send and receive operations.

12915 SHUT_WR Disables further send operations.

12916 The <sys/socket.h> header shall define the **size_t** and **ssize_t** types as described in

12917 <sys/types.h>.

12918 The following shall be declared as functions and may also be defined as macros. Function

12919 prototypes shall be provided.

```

12920 int      accept(int, struct sockaddr *restrict, socklen_t *restrict);
12921 int      bind(int, const struct sockaddr *, socklen_t);
12922 int      connect(int, const struct sockaddr *, socklen_t);
12923 int      getpeername(int, struct sockaddr *restrict, socklen_t *restrict);
12924 int      getsockname(int, struct sockaddr *restrict, socklen_t *restrict);
12925 int      getsockopt(int, int, int, void *restrict, socklen_t *restrict);
12926 int      listen(int, int);
12927 ssize_t  recv(int, void *, size_t, int);
12928 ssize_t  recvfrom(int, void *restrict, size_t, int,
12929                 struct sockaddr *restrict, socklen_t *restrict);
12930 ssize_t  recvmsg(int, struct msghdr *, int);
12931 ssize_t  send(int, const void *, size_t, int);
12932 ssize_t  sendmsg(int, const struct msghdr *, int);
12933 ssize_t  sendto(int, const void *, size_t, int, const struct sockaddr *,
12934                socklen_t);
12935 int      setsockopt(int, int, int, const void *, socklen_t);
12936 int      shutdown(int, int);
12937 int      socketatmark(int);
12938 int      socket(int, int, int);
12939 int      socketpair(int, int, int, int [2]);

```

12940 Inclusion of <sys/socket.h> may also make visible all symbols from <sys/uio.h>.

APPLICATION USAGE

12941 To forestall portability problems, it is recommended that applications not use values larger than

12942 $2^{31} - 1$ for the **socklen_t** type.

12943

12944 The **sockaddr_storage** structure solves the problem of declaring storage for automatic variables

12945 which is both large enough and aligned enough for storing the socket address data structure of

12946 any family. For example, code with a file descriptor and without the context of the address

12947 family can pass a pointer to a variable of this type, where a pointer to a socket address structure

12948 is expected in calls such as *getpeername()*, and determine the address family by accessing the

12949 received content after the call.

12950 The example below illustrates a data structure which aligns on a 64-bit boundary. An

12951 implementation-defined field *_ss_align* following *_ss_pad1* is used to force a 64-bit alignment

12952 which covers proper alignment good enough for needs of at least **sockaddr_in6** (IPv6) and

12953 **sockaddr_in** (IPv4) address data structures. The size of padding field *_ss_pad1* depends on the

12954 chosen alignment boundary. The size of padding field *_ss_pad2* depends on the value of overall

12955 size chosen for the total size of the structure. This size and alignment are represented in the

12956 above example by implementation-defined (not required) constants *_SS_MAXSIZE* (chosen

<sys/socket.h>

Headers

value 128) and `_SS_ALIGNMENT` (with chosen value 8). Constants `_SS_PAD1SIZE` (derived value 6) and `_SS_PAD2SIZE` (derived value 112) are also for illustration and not required. The implementation-defined definitions and structure field names above start with an <underscore> to denote implementation private name space. Portable code is not expected to access or reference those fields or constants.

```

12957
12958
12959
12960
12961
12962
12963
12964
12965
12966
12967
12968
12969
12970
12971
12972
12973
12974
12975
12976
12977
12978
12979
12980
12981
12982
12983
12984
12985
12986
12987
12988
12989
12990
12991
12992
12993
12994
12995
12996
12997
12998
12999
13000
13001
13002
13003

```

```

/*
 * Desired design of maximum size and alignment.
 */
#define _SS_MAXSIZE 128
/* Implementation-defined maximum size. */
#define _SS_ALIGNSIZE (sizeof(int64_t))
/* Implementation-defined desired alignment. */

/*
 * Definitions used for sockaddr_storage structure paddings design.
 */
#define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
#define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t) + \
    _SS_PAD1SIZE + _SS_ALIGNSIZE))
struct sockaddr_storage {
    sa_family_t ss_family; /* Address family. */
/*
 * Following fields are implementation-defined.
 */
    char _ss_pad1[_SS_PAD1SIZE];
    /* 6-byte pad; this is to make implementation-defined
       pad up to alignment field that follows explicit in
       the data structure. */
    int64_t _ss_align; /* Field to force desired structure
       storage alignment. */
    char _ss_pad2[_SS_PAD2SIZE];
    /* 112-byte pad to achieve desired size,
       _SS_MAXSIZE value minus size of ss_family
       __ss_pad1, __ss_align fields is 112. */
};

```

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
<sys/types.h>, <sys/uio.h>

XSH *accept()*, *bind()*, *connect()*, *getpeername()*, *getsockname()*, *getsockopt()*, *listen()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendmsg()*, *sendto()*, *setsockopt()*, *shutdown()*, *socketatmark()*, *socket()*, *socketpair()*

CHANGE HISTORY
First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the prototypes for *accept()*, *getpeername()*, *getsockname()*, *getsockopt()*, and *recvfrom()*.

*Headers***<sys/socket.h>**13004 **Issue 7**

13005 SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **ssize_t** type.

13006 SD5-XBD-ERN-62 is applied.

13007 The MSG_NOSIGNAL symbolic constant is added from The Open Group Technical Standard,
13008 2006, Extended API Set Part 2.

13009 This reference page is clarified with respect to macros and symbolic constants, and a declaration
13010 for the **size_t** type is added.

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

<sys/stat.h>

Headers

13011 NAME

13012 sys/stat.h — data returned by the stat() function

13013 SYNOPSIS

13014 #include <sys/stat.h>

13015 DESCRIPTION

13016 The <sys/stat.h> header shall define the structure of the data returned by the *fstat()*, *lstat()*, and
13017 *stat()* functions.

13018 The <sys/stat.h> header shall define the **stat** structure, which shall include at least the following
13019 members:

13020	dev_t st_dev	Device ID of device containing file.
13021	ino_t st_ino	File serial number.
13022	mode_t st_mode	Mode of file (see below).
13023	nlink_t st_nlink	Number of hard links to the file.
13024	uid_t st_uid	User ID of file.
13025	gid_t st_gid	Group ID of file.
13026	XSI dev_t st_rdev	Device ID (if file is character or block special).
13027	off_t st_size	For regular files, the file size in bytes.
13028		For symbolic links, the length in bytes of the
13029		pathname contained in the symbolic link.
13030	SHM	For a shared memory object, the length in bytes.
13031	TYM	For a typed memory object, the length in bytes.
13032		For other file types, the use of this field is
13033		unspecified.
13034	struct timespec st_atim	Last data access timestamp.
13035	struct timespec st_mtim	Last data modification timestamp.
13036	struct timespec st_ctim	Last file status change timestamp.
13037	XSI blksize_t st_blksize	A file system-specific preferred I/O block size
13038		for this object. In some file system types, this
13039		may vary from file to file.
13040	blkcnt_t st_blocks	Number of blocks allocated for this object.

13041 The *st_ino* and *st_dev* fields taken together uniquely identify the file within the system.

13042 The <sys/stat.h> header shall define the **blkcnt_t**, **blksize_t**, **dev_t**, **ino_t**, **mode_t**, **nlink_t**,
13043 **uid_t**, **gid_t**, **off_t**, and **time_t** types as described in <sys/types.h>.

13044 The <sys/stat.h> header shall define the **timespec** structure as described in <time.h>. Times
13045 shall be given in seconds since the Epoch.

13046 Which structure members have meaningful values depends on the type of file. For further
13047 information, see the descriptions of *fstat()*, *lstat()*, and *stat()* in the System Interfaces volume of
13048 POSIX.1-2008.

13049 For compatibility with earlier versions of this standard, the *st_atime* macro shall be defined with
13050 the value *st_atim.tv_sec*. Similarly, *st_ctime* and *st_mtime* shall be defined as macros with the
13051 values *st_ctim.tv_sec* and *st_mtim.tv_sec*, respectively.

Headers

<sys/stat.h>

The <sys/stat.h> header shall define the following symbolic constants for the file types encoded in type **mode_t**. The values shall be suitable for use in **#if** preprocessing directives:

S_IFMT Type of file.

S_IFBLK Block special.

S_IFCHR Character special.

S_IFIFO FIFO special.

S_IFREG Regular.

S_IFDIR Directory.

S_IFLNK Symbolic link.

S_IFSOCK Socket.

The <sys/stat.h> header shall define the following symbolic constants for the file mode bits encoded in type **mode_t**, with the indicated numeric values. These macros shall expand to an expression which has a type that allows them to be used, either singly or OR'ed together, as the third argument to *open()* without the need for a **mode_t** cast. The values shall be suitable for use in **#if** preprocessing directives.

Name	Numeric Value	Description
S_IRWXU	0700	Read, write, execute/search by owner.
S_IRUSR	0400	Read permission, owner.
S_IWUSR	0200	Write permission, owner.
S_IXUSR	0100	Execute/search permission, owner.
S_IRWXG	070	Read, write, execute/search by group.
S_IRGRP	040	Read permission, group.
S_IWGRP	020	Write permission, group.
S_IXGRP	010	Execute/search permission, group.
S_IRWXO	07	Read, write, execute/search by others.
S_IROTH	04	Read permission, others.
S_IWOTH	02	Write permission, others.
S_IXOTH	01	Execute/search permission, others.
S_ISUID	04000	Set-user-ID on execution.
S_ISGID	02000	Set-group-ID on execution.
S_ISVTX	01000	On directories, restricted deletion flag.

The following macros shall be provided to test whether a file is of the specified type. The value *m* supplied to the macros is the value of *st_mode* from a **stat** structure. The macro shall evaluate to a non-zero value if the test is true; 0 if the test is false.

S_ISBLK(*m*) Test for a block special file.

S_ISCHR(*m*) Test for a character special file.

S_ISDIR(*m*) Test for a directory.

S_ISFIFO(*m*) Test for a pipe or FIFO special file.

S_ISREG(*m*) Test for a regular file.

<sys/stat.h>

Headers

13091		<code>S_ISLNK(<i>m</i>)</code>	Test for a symbolic link.
13092		<code>S_ISSOCK(<i>m</i>)</code>	Test for a socket.
13093		The implementation may implement message queues, semaphores, or shared memory objects as distinct file types. The following macros shall be provided to test whether a file is of the specified type. The value of the <i>buf</i> argument supplied to the macros is a pointer to a stat structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the stat structure referenced by <i>buf</i> . Otherwise, the macro shall evaluate to zero.	
13094			
13095			
13096			
13097			
13098			
13099		<code>S_TYPEISMQ(<i>buf</i>)</code>	Test for a message queue.
13100		<code>S_TYPEISSEM(<i>buf</i>)</code>	Test for a semaphore.
13101		<code>S_TYPEISSHM(<i>buf</i>)</code>	Test for a shared memory object.
13102	TYM	The implementation may implement typed memory objects as distinct file types, and the following macro shall test whether a file is of the specified type. The value of the <i>buf</i> argument supplied to the macros is a pointer to a stat structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the stat structure referenced by <i>buf</i> . Otherwise, the macro shall evaluate to zero.	
13103			
13104			
13105			
13106			
13107		<code>S_TYPEISTMO(<i>buf</i>)</code>	Test macro for a typed memory object.
13108		The <sys/stat.h> header shall define the following symbolic constants as distinct integer values outside of the range [0,999 999 999], for use with the <i>futimens()</i> and <i>utimensat()</i> functions:	
13109			
13110		<code>UTIME_NOW</code>	
13111		<code>UTIME_OMIT</code>	
13112		The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.	
13113			
13114		<code>int chmod(const char *, mode_t);</code>	
13115		<code>int fchmod(int, mode_t);</code>	
13116		<code>int fchmodat(int, const char *, mode_t, int);</code>	
13117		<code>int fstat(int, struct stat *);</code>	
13118		<code>int fstatat(int, const char *restrict, struct stat *restrict, int);</code>	
13119		<code>int futimens(int, const struct timespec [2]);</code>	
13120		<code>int lstat(const char *restrict, struct stat *restrict);</code>	
13121		<code>int mkdir(const char *, mode_t);</code>	
13122		<code>int mkdirat(int, const char *, mode_t);</code>	
13123		<code>int mkfifo(const char *, mode_t);</code>	
13124		<code>int mkfifoat(int, const char *, mode_t);</code>	
13125	XSI	<code>int mknod(const char *, mode_t, dev_t);</code>	
13126		<code>int mknodat(int, const char *, mode_t, dev_t);</code>	
13127		<code>int stat(const char *restrict, struct stat *restrict);</code>	
13128		<code>mode_t umask(mode_t);</code>	
13129		<code>int utimensat(int, const char *, const struct timespec [2], int);</code>	

APPLICATION USAGE

Use of the macros is recommended for determining the type of a file.

RATIONALE

A conforming C-language application must include **<sys/stat.h>** for functions that have arguments or return values of type **mode_t**, so that symbolic values for that type can be used. An alternative would be to require that these constants are also defined by including **<sys/types.h>**.

The **S_ISUID** and **S_ISGID** bits may be cleared on any write, not just on *open()*, as some historical implementations do.

System calls that update the time entry fields in the **stat** structure must be documented by the implementors. POSIX-conforming systems should not update the time entry fields for functions listed in the System Interfaces volume of POSIX.1-2008 unless the standard requires that they do, except in the case of documented extensions to the standard.

Upon assignment, file timestamps are immediately converted to the resolution of the file system by truncation (i.e., the recorded time can be older than the actual time). For example, if the file system resolution is 1 microsecond, then a conforming *stat()* must always return an *st_mtim.tv_nsec* that is a multiple of 1000. Some older implementations returned higher-resolution timestamps while the *inode* information was cached, and then spontaneously truncated the *tv_nsec* fields when they were stored to and retrieved from disk, but this behavior does not conform.

Note that *st_dev* must be unique within a Local Area Network (LAN) in a “system” made up of multiple computers’ file systems connected by a LAN.

Networked implementations of a POSIX-conforming system must guarantee that all files visible within the file tree (including parts of the tree that may be remotely mounted from other machines on the network) on each individual processor are uniquely identified by the combination of the *st_ino* and *st_dev* fields.

The unit for the *st_blocks* member of the **stat** structure is not defined within POSIX.1-2008. In some implementations it is 512 bytes. It may differ on a file system basis. There is no correlation between values of the *st_blocks* and *st_blksize*, and the *f_bsize* (from **<sys/statvfs.h>**) structure members.

Traditionally, some implementations defined the multiplier for *st_blocks* in **<sys/param.h>** as the symbol **DEV_BSIZE**.

Some earlier versions of this standard did not specify values for the file mode bit macros. The expectation was that some implementors might choose to use a different encoding for these bits than the traditional one, and that new applications would use symbolic file modes instead of numeric. This version of the standard specifies the traditional encoding, in recognition that nearly 20 years after the first publication of this standard numeric file modes are still in widespread use by application developers, and that all conforming implementations still use the traditional encoding.

FUTURE DIRECTIONS

No new **S_IFMT** symbolic names for the file type values of **mode_t** will be defined by POSIX.1-2008; if new file types are required, they will only be testable through **S_ISxx()** or **S_TYPEISxxx()** macros instead.

<sys/stat.h>

Headers

SEE ALSO

<sys/statvfs.h>, <sys/types.h>, <time.h>

XSH *chmod()*, *fchmod()*, *fstat()*, *fstatat()*, *futimens()*, *mkdir()*, *mkfifo()*, *mknod()*, *umask()*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

The type of *st_blksize* is changed from **long** to **blksize_t**; the type of *st_blocks* is changed from **long** to **blkcnt_t**.

Issue 6

The *S_TYPEISMQ()*, *S_TYPEISSEM()*, and *S_TYPEISSHM()* macros are unconditionally mandated.

The Open Group Corrigendum U035/4 is applied. In the DESCRIPTION, the types **blksize_t** and **blkcnt_t** have been described.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types are mandated.

S_IFSOCK and *S_ISSOCK* are added for sockets.

The description of **stat** structure members is changed to reflect contents when file type is a symbolic link.

The test macro *S_TYPEISTMO* is added for alignment with IEEE Std 1003.1j-2000.

The **restrict** keyword is added to the prototypes for *lstat()* and *stat()*.

The *lstat()* function is made mandatory.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/17 is applied, adding text regarding the *st_blocks* member of the **stat** structure to the RATIONALE.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/25 is applied, adding to the DESCRIPTION that the **timespec** structure may be defined as described in the <time.h> header.

Issue 7

SD5-XSH-ERN-161 is applied, updating the DESCRIPTION to clarify that the descriptions of the interfaces should be consulted in order to determine which structure members have meaningful values.

The *fchmodat()*, *fstatat()*, *mkdirat()*, *mkfifoat()*, *mknodat()*, and *utimensat()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The *futimens()* function is added.

This reference page is clarified with respect to macros and symbolic constants.

Changes are made related to support for finegrained timestamps and the *UTIME_NOW* and *UTIME_OMIT* symbolic constants are added.

Headers

<sys/statvfs.h>

NAME

sys/statvfs.h — VFS File System information structure

SYNOPSIS

```
#include <sys/statvfs.h>
```

DESCRIPTION

The <sys/statvfs.h> header shall define the **statvfs** structure, which shall include at least the following members:

unsigned long	f_bsize	File system block size.
unsigned long	f_frsize	Fundamental file system block size.
fsblkcnt_t	f_blocks	Total number of blocks on file system in units of <i>f_frsize</i> .
fsblkcnt_t	f_bfree	Total number of free blocks.
fsblkcnt_t	f_bavail	Number of free blocks available to non-privileged process.
fsfilcnt_t	f_files	Total number of file serial numbers.
fsfilcnt_t	f_ffree	Total number of free file serial numbers.
fsfilcnt_t	f_favail	Number of file serial numbers available to non-privileged process.
unsigned long	f_fsid	File system ID.
unsigned long	f_flag	Bit mask of <i>f_flag</i> values.
unsigned long	f_namemax	Maximum filename length.

The <sys/statvfs.h> header shall define the **fsblkcnt_t** and **fsfilcnt_t** types as described in <sys/types.h>.

The <sys/statvfs.h> header shall define the following symbolic constants for the *f_flag* member:

ST_RDONLY Read-only file system.

ST_NOSUID Does not support the semantics of the ST_ISUID and ST_ISGID file mode bits.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int fstatvfs(int, struct statvfs *);
int statvfs(const char *restrict, struct statvfs *restrict);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *fstatvfs()*

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

The type of *f_blocks*, *f_bfree*, and *f_bavail* is changed from **unsigned long** to **fsblkcnt_t**; the type of *f_files*, *f_ffree*, and *f_favail* is changed from **unsigned long** to **fsfilcnt_t**.

<sys/statvfs.h>

Headers

13253 **Issue 6**

13254 The Open Group Corrigendum U035/5 is applied. In the DESCRIPTION, the types **fsblkcnt_t**
13255 and **fsfilcnt_t** have been described.

13256 The **restrict** keyword is added to the prototype for *statvfs()*.

13257 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/18 is applied, changing the description of
13258 ST_NOSUID from “Does not support *setuid()*/*setgid()* semantics” to “Does not support the
13259 semantics of the ST_ISUID and ST_ISGID file mode bits”.

13260 **Issue 7**

13261 The <sys/statvfs.h> header is moved from the XSI option to the Base.

13262 This reference page is clarified with respect to macros and symbolic constants.

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

Headers

<sys/time.h>

NAME

sys/time.h — time types

SYNOPSIS

```
#include <sys/time.h>
```

DESCRIPTION

The <sys/time.h> header shall define the **timeval** structure, which shall include at least the following members:

time_t	tv_sec	Seconds.
suseconds_t	tv_usec	Microseconds.

The <sys/time.h> header shall define the **itimerval** structure, which shall include at least the following members:

struct timeval	it_interval	Timer interval.
struct timeval	it_value	Current value.

The <sys/time.h> header shall define the **time_t** and **suseconds_t** types as described in <sys/types.h>.

The <sys/time.h> header shall define the **fd_set** type as described in <sys/select.h>.

The <sys/time.h> header shall define the following symbolic constants for the *which* argument of *getitimer()* and *setitimer()*:

ITIMER_REAL	Decrements in real time.
ITIMER_VIRTUAL	Decrements in process virtual time.
ITIMER_PROF	Decrements both in process virtual time and when the system is running on behalf of the process.

The <sys/time.h> header shall define the following as described in <sys/select.h>:

```
FD_CLR()
FD_ISSET()
FD_SET()
FD_ZERO()
FD_SETSIZE
```

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int getitimer(int, struct itimerval *);
int gettimeofday(struct timeval *restrict, void *restrict);
int setitimer(int, const struct itimerval *restrict,
              struct itimerval *restrict);
int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
           struct timeval *restrict);
int utimes(const char *, const struct timeval [2]);
```

Inclusion of the <sys/time.h> header may make visible all symbols from the <sys/select.h> header.

<sys/time.h>

Headers

13302 APPLICATION USAGE

13303 None.

13304 RATIONALE

13305 None.

13306 FUTURE DIRECTIONS

13307 None.

13308 SEE ALSO

13309 <sys/select.h>, <sys/types.h>

13310 XSH *futimens()*, *getitimer()*, *gettimeofday()*, *pselect()*

13311 CHANGE HISTORY

13312 First released in Issue 4, Version 2.

13313 Issue 5

13314 The type of *tv_usec* is changed from **long** to **suseconds_t**.

13315 Issue 6

13316 The **restrict** keyword is added to the prototypes for *gettimeofday()*, *select()*, and *setitimer()*.13317 The note is added that inclusion of this header may also make symbols visible from
13318 <sys/select.h>.13319 The *utimes()* function is marked LEGACY.

13320 Issue 7

13321 This reference page is clarified with respect to macros and symbolic constants.

Headers

<sys/times.h>

NAME

sys/times.h — file access and modification times structure

SYNOPSIS

```
#include <sys/times.h>
```

DESCRIPTION

The <sys/times.h> header shall define the **tms** structure, which is returned by *times()* and shall include at least the following members:

```
clock_t  tms_utime  User CPU time.
clock_t  tms_stime  System CPU time.
clock_t  tms_cutime User CPU time of terminated child processes.
clock_t  tms_cstime System CPU time of terminated child processes.
```

The <sys/times.h> header shall define the **clock_t** type as described in <sys/types.h>.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
clock_t times(struct tms *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>

XSH *times()*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

<sys/types.h>

Headers

13348 NAME

13349 sys/types.h — data types

13350 SYNOPSIS

13351 #include <sys/types.h>

13352 DESCRIPTION

13353 The <sys/types.h> header shall define at least the following types:

13354	blkcnt_t	Used for file block counts.
13355	blksize_t	Used for block sizes.
13356	clock_t	Used for system times in clock ticks or CLOCKS_PER_SEC; see <time.h>.
13357		
13358	clockid_t	Used for clock ID type in the clock and timer functions.
13359	dev_t	Used for device IDs.
13360	XSI fsblkcnt_t	Used for file system block counts.
13361	XSI fsfilcnt_t	Used for file system file counts.
13362	gid_t	Used for group IDs.
13363	id_t	Used as a general identifier; can be used to contain at least a pid_t ,
13364		uid_t , or gid_t .
13365	ino_t	Used for file serial numbers.
13366	XSI key_t	Used for XSI interprocess communication.
13367	mode_t	Used for some file attributes.
13368	nlink_t	Used for link counts.
13369	off_t	Used for file sizes.
13370	pid_t	Used for process IDs and process group IDs.
13371	pthread_attr_t	Used to identify a thread attribute object.
13372	pthread_barrier_t	Used to identify a barrier.
13373	pthread_barrierattr_t	Used to define a barrier attributes object.
13374	pthread_cond_t	Used for condition variables.
13375	pthread_condattr_t	Used to identify a condition attribute object.
13376	pthread_key_t	Used for thread-specific data keys.
13377	pthread_mutex_t	Used for mutexes.
13378	pthread_mutexattr_t	Used to identify a mutex attribute object.
13379	pthread_once_t	Used for dynamic package initialization.
13380	pthread_rwlock_t	Used for read-write locks.
13381	pthread_rwlockattr_t	Used for read-write lock attributes.
13382	pthread_spinlock_t	Used to identify a spin lock.

Headers

<sys/types.h>

13383		pthread_t	Used to identify a thread.
13384		size_t	Used for sizes of objects.
13385		ssize_t	Used for a count of bytes or an error indication.
13386	XSI	suseconds_t	Used for time in microseconds.
13387		time_t	Used for time in seconds.
13388		timer_t	Used for timer ID returned by <i>timer_create()</i> .
13389	OB TRC		Also used to identify a trace stream attributes object.
13390	OB TRC	trace_event_id_t	Used to identify a trace event type.
13391	OB TEF	trace_event_set_t	Used to identify a trace event type set.
13392	OB TRC	trace_id_t	Used to identify a trace stream.
13393		uid_t	Used for user IDs.
13394		All of the types shall be defined as arithmetic types of an appropriate length, with the following exceptions:	
13395			
13396		pthread_attr_t	
13397		pthread_barrier_t	
13398		pthread_barrierattr_t	
13399		pthread_cond_t	
13400		pthread_condattr_t	
13401		pthread_key_t	
13402		pthread_mutex_t	
13403		pthread_mutexattr_t	
13404		pthread_once_t	
13405		pthread_rwlock_t	
13406		pthread_rwlockattr_t	
13407		pthread_spinlock_t	
13408		pthread_t	
13409	OB TRC	trace_attr_t	
13410		trace_event_id_t	
13411	OB TEF	trace_event_set_t	
13412	OB TRC	trace_id_t	
13413		Additionally:	
13414		• mode_t shall be an integer type.	
13415		• nlink_t , uid_t , gid_t , and id_t shall be integer types.	
13416		• blkcnt_t and off_t shall be signed integer types.	
13417	XSI	• fsblkcnt_t , fsfilcnt_t , and ino_t shall be defined as unsigned integer types.	
13418		• size_t shall be an unsigned integer type.	
13419		• blksize_t , pid_t , and ssize_t shall be signed integer types.	
13420		• time_t and clock_t shall be integer or real-floating types.	
13421		The type ssize_t shall be capable of storing values at least in the range $[-1, \{SSIZE\_MAX\}]$.	

<sys/types.h>

Headers

13422 XSI The type **suseconds_t** shall be a signed integer type capable of storing values at least in the
 13423 range $[-1, 1\,000\,000]$.

13424 The implementation shall support one or more programming environments in which the widths
 13425 of **blksize_t**, **pid_t**, **size_t**, **ssize_t**, and **suseconds_t** are no greater than the width of type **long**.
 13426 The names of these programming environments can be obtained using the *confstr()* function or
 13427 the *getconf* utility.

13428 There are no defined comparison or assignment operators for the following types:

13429 **pthread_attr_t**
 13430 **pthread_barrier_t**
 13431 **pthread_barrierattr_t**
 13432 **pthread_cond_t**
 13433 **pthread_condattr_t**
 13434 **pthread_mutex_t**
 13435 **pthread_mutexattr_t**
 13436 **pthread_rwlock_t**
 13437 **pthread_rwlockattr_t**
 13438 **pthread_spinlock_t**
 13439 OB TRC **trace_attr_t**

13440 APPLICATION USAGE

13441 None.

13442 RATIONALE

13443 None.

13444 FUTURE DIRECTIONS

13445 None.

13446 SEE ALSO

13447 [<time.h>](#)13448 XSH *confstr()*13449 XCU *getconf*

13450 CHANGE HISTORY

13451 First released in Issue 1. Derived from Issue 1 of the SVID.

13452 Issue 5

13453 The **clockid_t** and **timer_t** types are defined for alignment with the POSIX Realtime Extension.13454 The types **blkcnt_t**, **blksize_t**, **fsblkcnt_t**, **fsfilcnt_t**, and **suseconds_t** are added.

13455 Large File System extensions are added.

13456 Updated for alignment with the POSIX Threads Extension.

13457 Issue 6

13458 The **pthread_barrier_t**, **pthread_barrierattr_t**, and **pthread_spinlock_t** types are added for
 13459 alignment with IEEE Std 1003.1j-2000.13460 The margin code is changed from XSI to THR for the **pthread_rwlock_t** and
 13461 **pthread_rwlockattr_t** types as Read-Write Locks have been absorbed into the POSIX Threads
 13462 option. The threads types are marked THR.

Headers

<sys/types.h>

- 13463 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/26 is applied, adding **pthread_t** to the list
13464 of types that are not required to be arithmetic types, thus allowing **pthread_t** to be defined as a
13465 structure.
- 13466 **Issue 7**
- 13467 Austin Group Interpretation 1003.1-2001 #033 is applied, requiring **key_t** to be an arithmetic
13468 type.
- 13469 The Trace option types are marked obsolescent.
- 13470 The **clock_t** and **id_t** types are moved from the XSI option to the Base.
- 13471 The **pthread_barrier_t** and **pthread_barrierattr_t** types are moved from the Barriers option to
13472 the Base.
- 13473 The **pthread_spinlock_t** type is moved from the Spin Locks option to the Base.
- 13474 Functionality relating to the Timers and Threads options is moved to the Base.

<sys/uio.h>*Headers*13475 **NAME**

13476 sys/uio.h — definitions for vector I/O operations

13477 **SYNOPSIS**13478 XSI `#include <sys/uio.h>`13479 **DESCRIPTION**13480 The **<sys/uio.h>** header shall define the **iovec** structure, which shall include at least the
13481 following members:13482 `void *iov_base` Base address of a memory region for input or output.13483 `size_t iov_len` The size of the memory pointed to by *iov_base*.13484 The **<sys/uio.h>** header uses the **iovec** structure for scatter/gather I/O.13485 The **<sys/uio.h>** header shall define the **ssize_t** and **size_t** types as described in **<sys/types.h>**.13486 The following shall be declared as functions and may also be defined as macros. Function
13487 prototypes shall be provided.13488 `ssize_t readv(int, const struct iovec *, int);`13489 `ssize_t writev(int, const struct iovec *, int);`13490 **APPLICATION USAGE**13491 The implementation can put a limit on the number of scatter/gather elements which can be
13492 processed in one call. The symbol {IOV_MAX} defined in **<limits.h>** should always be used to
13493 learn about the limits instead of assuming a fixed value.13494 **RATIONALE**13495 Traditionally, the maximum number of scatter/gather elements the system can process in one
13496 call were described by the symbolic value {UIO_MAXIOV}. In IEEE Std 1003.1-2001 this value is
13497 replaced by the constant {IOV_MAX} which can be found in **<limits.h>**.13498 **FUTURE DIRECTIONS**

13499 None.

13500 **SEE ALSO**13501 **<limits.h>**, **<sys/types.h>**13502 XSH *read()*, *readv()*, *write()*, *writev()*13503 **CHANGE HISTORY**

13504 First released in Issue 4, Version 2.

13505 **Issue 6**

13506 Text referring to scatter/gather I/O is added to the DESCRIPTION.

Headers

<sys/un.h>

NAME

sys/un.h — definitions for UNIX domain sockets

SYNOPSIS

```
#include <sys/un.h>
```

DESCRIPTION

The <sys/un.h> header shall define the **sockaddr_un** structure, which shall include at least the following members:

```
sa_family_t  sun_family  Address family.
char         sun_path[]  Socket pathname.
```

The **sockaddr_un** structure is used to store addresses for UNIX domain sockets. Values of this type shall be cast by applications to **struct sockaddr** for use with socket functions.

The <sys/un.h> header shall define the **sa_family_t** type as described in <sys/socket.h>.

APPLICATION USAGE

The size of *sun_path* has intentionally been left undefined. This is because different implementations use different sizes. For example, 4.3 BSD uses a size of 108, and 4.4 BSD uses a size of 104. Since most implementations originate from BSD versions, the size is typically in the range 92 to 108.

Applications should not assume a particular length for *sun_path* or assume that it can hold {_POSIX_PATH_MAX} bytes (256).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/socket.h>

XSH *bind()*, *socket()*, *socketpair()*

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

The value for {_POSIX_PATH_MAX} is updated to 256.

<sys/utsname.h>

Headers

13537 NAME

13538 sys/utsname.h — system name structure

13539 SYNOPSIS

13540 #include <sys/utsname.h>

13541 DESCRIPTION

13542 The <sys/utsname.h> header shall define the structure **utsname** which shall include at least the
 13543 following members:

13544 char sysname[] Name of this implementation of the operating system.
 13545 char nodename[] Name of this node within the communications
 13546 network to which this node is attached, if any.
 13547 char release[] Current release level of this implementation.
 13548 char version[] Current version level of this release.
 13549 char machine[] Name of the hardware type on which the system is running.

13550 The character arrays are of unspecified size, but the data stored in them shall be terminated by a
 13551 null byte.

13552 The following shall be declared as a function and may also be defined as a macro:

13553 int uname(struct utsname *);

13554 APPLICATION USAGE

13555 None.

13556 RATIONALE

13557 None.

13558 FUTURE DIRECTIONS

13559 None.

13560 SEE ALSO

13561 XSH *uname()*

13562 CHANGE HISTORY

13563 First released in Issue 1. Derived from Issue 1 of the SVID.

13564 Issue 6

13565 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/27 is applied, changing the description of
 13566 *nodename* within the **utsname** structure from “an implementation-defined communications
 13567 network” to “the communications network to which this node is attached, if any”.

Headers

<sys/wait.h>

NAME

sys/wait.h — declarations for waiting

SYNOPSIS

```
#include <sys/wait.h>
```

DESCRIPTION

The <sys/wait.h> header shall define the following symbolic constants for use with *waitpid()*:

XSI	WCONTINUED	Report status of continued child process.
	WNOHANG	Do not hang if no status is available; return immediately.
	WUNTRACED	Report status of stopped child process.

The <sys/wait.h> header shall define the following macros for analysis of process status values:

	WEXITSTATUS	Return exit status.
XSI	WIFCONTINUED	True if child has been continued.
	WIFEXITED	True if child exited normally.
	WIFSIGNALED	True if child exited due to uncaught signal.
	WIFSTOPPED	True if child is currently stopped.
	WSTOPSIG	Return signal number that caused process to stop.
	WTERMSIG	Return signal number that caused process to terminate.

The <sys/wait.h> header shall define the following symbolic constants as possible values for the *options* argument to *waitid()*:

	WEXITED	Wait for processes that have exited.
	WNOWAIT	Keep the process whose status is returned in <i>infop</i> in a waitable state.
	WSTOPPED	Status is returned for any child that has stopped upon receipt of a signal.

The XSI **WCONTINUED** and **WNOHANG** constants, described above for *waitpid()*, can also be used with *waitid()*.

The type **idtype_t** shall be defined as an enumeration type whose possible values shall include at least the following:

```
P_ALL
P_PGID
P_PID
```

The <sys/wait.h> header shall define the **id_t** and **pid_t** types as described in <sys/types.h>.

The <sys/wait.h> header shall define the **siginfo_t** type as described in <signal.h>.

Inclusion of the <sys/wait.h> header may also make visible all symbols from <signal.h>.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
pid_t wait(int *);
int waitid(idtype_t, id_t, siginfo_t *, int);
pid_t waitpid(pid_t, int *, int);
```

<sys/wait.h>

Headers

13605 APPLICATION USAGE

13606 None.

13607 RATIONALE

13608 None.

13609 FUTURE DIRECTIONS

13610 None.

13611 SEE ALSO

13612 <signal.h>, <sys/types.h>

13613 XSH *wait()*, *waitid()*

13614 CHANGE HISTORY

13615 First released in Issue 3.

13616 Included for alignment with the POSIX.1-1988 standard.

13617 Issue 6

13618 The *wait3()* function is removed.

13619 Issue 7

13620 The *waitid()* function and symbolic constants for its *options* argument are moved to the Base.

13621 The description of the WNOHANG constant is clarified.

Headers

<syslog.h>

NAME

syslog.h — definitions for system error logging

SYNOPSIS

```
#include <syslog.h>
```

DESCRIPTION

The <syslog.h> header shall define the following symbolic constants, zero or more of which may be OR'ed together to form the *logopt* option of *openlog()*:

LOG_PID	Log the process ID with each message.
LOG_CONS	Log to the system console on error.
LOG_NDELAY	Connect to syslog daemon immediately.
LOG_ODELAY	Delay open until <i>syslog()</i> is called.
LOG_NOWAIT	Do not wait for child processes.

The <syslog.h> header shall define the following symbolic constants for use as the *facility* argument to *openlog()*:

LOG_KERN	Reserved for message generated by the system.
LOG_USER	Message generated by a process.
LOG_MAIL	Reserved for message generated by mail system.
LOG_NEWS	Reserved for message generated by news system.
LOG_UUCP	Reserved for message generated by UUCP system.
LOG_DAEMON	Reserved for message generated by system daemon.
LOG_AUTH	Reserved for message generated by authorization daemon.
LOG_CRON	Reserved for message generated by clock daemon.
LOG_LPR	Reserved for message generated by printer system.
LOG_LOCAL0	Reserved for local use.
LOG_LOCAL1	Reserved for local use.
LOG_LOCAL2	Reserved for local use.
LOG_LOCAL3	Reserved for local use.
LOG_LOCAL4	Reserved for local use.
LOG_LOCAL5	Reserved for local use.
LOG_LOCAL6	Reserved for local use.
LOG_LOCAL7	Reserved for local use.

The <syslog.h> header shall define the following macros for constructing the *maskpri* argument to *setlogmask()*. The following macros expand to an expression of type **int** when the argument *pri* is an expression of type **int**:

LOG_MASK(<i>pri</i>)	A mask for priority <i>pri</i> .

The <syslog.h> header shall define the following symbolic constants for use as the *priority* argument of *syslog()*:

<syslog.h>

Headers

13659	LOG_EMERG	A panic condition was reported to all processes.
13660	LOG_ALERT	A condition that should be corrected immediately.
13661	LOG_CRIT	A critical condition.
13662	LOG_ERR	An error message.
13663	LOG_WARNING	A warning message.
13664	LOG_NOTICE	A condition requiring special handling.
13665	LOG_INFO	A general information message.
13666	LOG_DEBUG	A message useful for debugging programs.
13667	The following shall be declared as functions and may also be defined as macros. Function	
13668	prototypes shall be provided.	
13669	void closelog(void);	
13670	void openlog(const char *, int, int);	
13671	int setlogmask(int);	
13672	void syslog(int, const char *, ...);	
13673	APPLICATION USAGE	
13674	None.	
13675	RATIONALE	
13676	None.	
13677	FUTURE DIRECTIONS	
13678	None.	
13679	SEE ALSO	
13680	XSH <i>closelog()</i>	
13681	CHANGE HISTORY	
13682	First released in Issue 4, Version 2.	
13683	Issue 5	
13684	Moved from X/Open UNIX to BASE.	
13685	Issue 7	
13686	This reference page is clarified with respect to macros and symbolic constants.	

*Headers***<tar.h>****NAME**

tar.h — extended tar definitions

SYNOPSIS

#include <tar.h>

DESCRIPTIONThe **<tar.h>** header shall define the following symbolic constants with the indicated values.

General definitions:

Name	Value	Description
TMAGIC	"ustar"	ustar plus null byte.
TMAGLEN	6	Length of the above.
TVERSION	"00"	00 without a null byte.
TVERSLEN	2	Length of the above.

Typeflag field definitions:

Name	Value	Description
REGTYPE	'0'	Regular file.
AREGTYPE	'\0'	Regular file.
LNKTYPE	'1'	Link.
SYMTYPE	'2'	Symbolic link.
CHRTYPE	'3'	Character special.
BLKTYPE	'4'	Block special.
DIRTYPE	'5'	Directory.
FIFOTYPE	'6'	FIFO special.
CONTTYPE	'7'	Reserved.

Mode field bit definitions (octal):

Name	Value	Description
TSUID	04000	Set UID on execution.
TSGID	02000	Set GID on execution.
TSVTX	01000	On directories, restricted deletion flag.
TUREAD	00400	Read by owner.
TUWRITE	00200	Write by owner special.
TUEXEC	00100	Execute/search by owner.
TGREAD	00040	Read by group.
TGWRITE	00020	Write by group.
TGEXEC	00010	Execute/search by group.
TOREAD	00004	Read by other.
TOWRITE	00002	Write by other.
TOEXEC	00001	Execute/search by other.

<tar.h>

Headers

13724 **APPLICATION USAGE**

13725 None.

13726 **RATIONALE**

13727 None.

13728 **FUTURE DIRECTIONS**

13729 None.

13730 **SEE ALSO**13731 XCU *pax*13732 **CHANGE HISTORY**

13733 First released in Issue 3. Derived from the POSIX.1-1988 standard.

13734 **Issue 6**13735 The SEE ALSO section is updated to refer to *pax*.13736 **Issue 7**

13737 This reference page is clarified with respect to macros and symbolic constants.

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

Headers

<termios.h>

NAME

termios.h — define values for termios

SYNOPSIS

```
#include <termios.h>
```

DESCRIPTION

The <termios.h> header shall contain the definitions used by the terminal I/O interfaces (see Chapter 11 (on page 199) for the structures and names defined).

The termios Structure

The <termios.h> header shall define the following data types through **typedef**:

cc_t Used for terminal special characters.

speed_t Used for terminal baud rates.

tcflag_t Used for terminal modes.

The above types shall be all unsigned integer types.

The implementation shall support one or more programming environments in which the widths of **cc_t**, **speed_t**, and **tcflag_t** are no greater than the width of type **long**. The names of these programming environments can be obtained using the *confstr()* function or the *getconf* utility.

The <termios.h> header shall define the **termios** structure, which shall include at least the following members:

tcflag_t	c_iflag	Input modes.
tcflag_t	c_oflag	Output modes.
tcflag_t	c_cflag	Control modes.
tcflag_t	c_lflag	Local modes.
cc_t	c_cc[NCCS]	Control characters.

The <termios.h> header shall define the following symbolic constant:

NCCS Size of the array *c_cc* for control characters.

The <termios.h> header shall define the following symbolic constants for use as subscripts for the array *c_cc*:

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF	VINTR	EOF character.
VEOL		EOL character.
VERASE		ERASE character.
VINTR		INTR character.
VKILL		KILL character.
	VMIN	MIN value.
VQUIT	VQUIT	QUIT character.
VSTART	VSTART	START character.
VSTOP	VSTOP	STOP character.
VSUSP	VSUSP	SUSP character.
	VTIME	TIME value.

The subscript values shall be suitable for use in **#if** preprocessing directives and shall be distinct, except that the VMIN and VTIME subscripts may have the same values as the VEOF and VEOL subscripts, respectively.

<termios.h>

Headers

13781 **Input Modes**

13782 The <termios.h> header shall define the following symbolic constants for use as flags in the
13783 *c_iflag* field. The *c_iflag* field describes the basic terminal input control.

13784	BRKINT	Signal interrupt on break.
13785	ICRNL	Map CR to NL on input.
13786	IGNBRK	Ignore break condition.
13787	IGNCR	Ignore CR.
13788	IGNPAR	Ignore characters with parity errors.
13789	INLCR	Map NL to CR on input.
13790	INPCK	Enable input parity check.
13791	ISTRIP	Strip character.
13792	IXANY	Enable any character to restart output.
13793	IXOFF	Enable start/stop input control.
13794	IXON	Enable start/stop output control.
13795	PARMRK	Mark parity errors.

13796 **Output Modes**

13797 The <termios.h> header shall define the following symbolic constants for use as flags in the
13798 *c_oflag* field. The *c_oflag* field specifies the system treatment of output.

13799	OPOST	Post-process output.
13800	XSI ONLCR	Map NL to CR-NL on output.
13801	XSI OCRNL	Map CR to NL on output.
13802	XSI ONOCR	No CR output at column 0.
13803	XSI ONLRET	NL performs CR function.
13804	XSI OFDEL	Fill is DEL.
13805	XSI OFILL	Use fill characters for delay.
13806	XSI NLDLY	Select newline delays:
13807		NL0 Newline type 0.
13808		NL1 Newline type 1.
13809	XSI CRDLY	Select carriage-return delays:
13810		CR0 Carriage-return delay type 0.
13811		CR1 Carriage-return delay type 1.
13812		CR2 Carriage-return delay type 2.
13813		CR3 Carriage-return delay type 3.

Headers

<termios.h>

13814	XSI	TABDLY	Select horizontal-tab delays:
13815		TAB0	Horizontal-tab delay type 0.
13816		TAB1	Horizontal-tab delay type 1.
13817		TAB2	Horizontal-tab delay type 2.
13818		TAB3	Expand tabs to spaces.
13819	XSI	BSDLY	Select backspace delays:
13820		BS0	Backspace-delay type 0.
13821		BS1	Backspace-delay type 1.
13822	XSI	VTDLY	Select vertical-tab delays:
13823		VT0	Vertical-tab delay type 0.
13824		VT1	Vertical-tab delay type 1.
13825	XSI	FFDLY	Select form-feed delays:
13826		FF0	Form-feed delay type 0.
13827		FF1	Form-feed delay type 1.

Baud Rate Selection

The <termios.h> header shall define the following symbolic constants for use as values of objects of type **speed_t**.

The input and output baud rates are stored in the **termios** structure. These are the valid values for objects of type **speed_t**. Not all baud rates need be supported by the underlying hardware.

13833	B0	Hang up
13834	B50	50 baud
13835	B75	75 baud
13836	B110	110 baud
13837	B134	134.5 baud
13838	B150	150 baud
13839	B200	200 baud
13840	B300	300 baud
13841	B600	600 baud
13842	B1200	1 200 baud
13843	B1800	1 800 baud
13844	B2400	2 400 baud

<termios.h>

Headers

13845	B4800	4 800 baud
13846	B9600	9 600 baud
13847	B19200	19 200 baud
13848	B38400	38 400 baud

13849 **Control Modes**

13850 The <termios.h> header shall define the following symbolic constants for use as flags in the
 13851 *c_cflag* field. The *c_cflag* field describes the hardware control of the terminal; not all values
 13852 specified are required to be supported by the underlying hardware.

13853	CSIZE	Character size:
13854		CS5 5 bits
13855		CS6 6 bits
13856		CS7 7 bits
13857		CS8 8 bits
13858	CSTOPB	Send two stop bits, else one.
13859	CREAD	Enable receiver.
13860	PARENB	Parity enable.
13861	PARODD	Odd parity, else even.
13862	HUPCL	Hang up on last close.
13863	CLOCAL	Ignore modem status lines.

13864 The implementation shall support the functionality associated with the symbols CS7, CS8,
 13865 CSTOPB, PARODD, and PARENB.

13866 **Local Modes**

13867 The <termios.h> header shall define the following symbolic constants for use as flags in the
 13868 *c_lflag* field. The *c_lflag* field of the argument structure is used to control various terminal
 13869 functions.

13870	ECHO	Enable echo.
13871	ECHOE	Echo erase character as error-correcting backspace.
13872	ECHOK	Echo KILL.
13873	ECHONL	Echo NL.
13874	ICANON	Canonical input (erase and kill processing).
13875	IEXTEN	Enable extended input character processing.
13876	ISIG	Enable signals.
13877	NOFLSH	Disable flush after interrupt or quit.
13878	TOSTOP	Send SIGTTOU for background output.

Attribute Selection

The <termios.h> header shall define the following symbolic constants for use with *tcsetattr()*:

TCSANOW	Change attributes immediately.
TCSADRAIN	Change attributes when output has drained.
TCSAFLUSH	Change attributes when output has drained; also flush pending input.

Line Control

The <termios.h> header shall define the following symbolic constants for use with *tcflush()*:

TCIFLUSH	Flush pending input.
TCIOFLUSH	Flush both pending input and untransmitted output.
TCOFLUSH	Flush untransmitted output.

The <termios.h> header shall define the following symbolic constants for use with *tcflow()*:

TCIOFF	Transmit a STOP character, intended to suspend input data.
TCION	Transmit a START character, intended to restart input data.
TCOOFF	Suspend output.
TCOON	Restart output.

The <termios.h> header shall define the **pid_t** type as described in <sys/types.h>.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```

speed_t cfgetispeed(const struct termios *);
speed_t cfgetospeed(const struct termios *);
int      cfsetispeed(struct termios *, speed_t);
int      cfsetospeed(struct termios *, speed_t);
int      tcdrain(int);
int      tcflow(int, int);
int      tcflush(int, int);
int      tcgetattr(int, struct termios *);
pid_t    tcgetsid(int);
int      tcsendbreak(int, int);
int      tcsetattr(int, int, const struct termios *);

```

APPLICATION USAGE

The following names are reserved for XSI-conformant systems to use as an extension to the above; therefore strictly conforming applications shall not use them:

CBAUD	EXTB	VDSUSP
DEFECHO	FLUSHO	VLNEXT
ECHOCTL	LOBLK	VREPRINT
ECHOKE	PENDIN	VSTATUS
ECHOPRT	SWTCH	VWERASE
EXTA	VDISCARD	

<termios.h>

Headers

13917 **RATIONALE**

13918 None.

13919 **FUTURE DIRECTIONS**

13920 None.

13921 **SEE ALSO**

13922 <sys/types.h>

13923 XSH *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *confstr()*, *tcdrain()*, *tcflow()*, *tcflush()*,
 13924 *tcgetattr()*, *tcgetsid()*, *tcsendbreak()*, *tcsetattr()*

13925 XCU Chapter 11 (on page 199), *getconf*13926 **CHANGE HISTORY**

13927 First released in Issue 3.

13928 Included for alignment with the ISO POSIX-1 standard.

13929 **Issue 6**

13930 The LEGACY symbols IUCLC, OLCUC, and XCASE are removed.

13931 FIPS 151-2 requirements for the symbols CS7, CS8, CSTOPB, PARODD, and PARENB are
 13932 reaffirmed.

13933 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/19 is applied, changing ECHOK to
 13934 ECHOK in the APPLICATION USAGE section.

13935 **Issue 7**

13936 Austin Group Interpretation 1003.1-2001 #144 is applied, moving functionality relating to the
 13937 IXANY symbol from the XSI option to the Base.

13938 SD5-XBD-ERN-35 is applied, adding the OFDEL output mode.

13939 This reference page is clarified with respect to macros and symbolic constants, and a declaration
 13940 for the **pid_t** type is added.

Headers

<tgmath.h>

NAME

tgmath.h — type-generic macros

SYNOPSIS

```
#include <tgmath.h>
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The <tgmath.h> header shall include the headers <math.h> and <complex.h> and shall define several type-generic macros.

Of the functions contained within the <math.h> and <complex.h> headers without an *f* (**float**) or *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is **double**. For each such function, except *modf()*, *j0()*, *j1()*, *jn()*, *y0()*, *y1()*, and *yn()*, there shall be a corresponding type-generic macro. The parameters whose corresponding real type is **double** in the function synopsis are generic parameters. Use of the macro invokes a function whose corresponding real type and type domain are determined by the arguments for the generic parameters.

Use of the macro invokes a function whose generic parameters have the corresponding real type determined as follows:

- First, if any argument for generic parameters has type **long double**, the type determined is **long double**.
- Otherwise, if any argument for generic parameters has type **double** or is of integer type, the type determined is **double**.
- Otherwise, the type determined is **float**.

For each unsuffixed function in the <math.h> header for which there is a function in the <complex.h> header with the same name except for a *c* prefix, the corresponding type-generic macro (for both functions) has the same name as the function in the <math.h> header. The corresponding type-generic macro for *fabs()* and *cabs()* is *fabs()*.

<math.h> Function	<complex.h> Function	Type-Generic Macro
<i>acos()</i>	<i>cacos()</i>	<i>acos()</i>
<i>asin()</i>	<i>casin()</i>	<i>asin()</i>
<i>atan()</i>	<i>catan()</i>	<i>atan()</i>
<i>acosh()</i>	<i>cacosh()</i>	<i>acosh()</i>
<i>asinh()</i>	<i>casinh()</i>	<i>asinh()</i>
<i>atanh()</i>	<i>catanh()</i>	<i>atanh()</i>
<i>cos()</i>	<i>ccos()</i>	<i>cos()</i>
<i>sin()</i>	<i>csin()</i>	<i>sin()</i>
<i>tan()</i>	<i>ctan()</i>	<i>tan()</i>
<i>cosh()</i>	<i>ccosh()</i>	<i>cosh()</i>
<i>sinh()</i>	<i>csinh()</i>	<i>sinh()</i>
<i>tanh()</i>	<i>ctanh()</i>	<i>tanh()</i>
<i>exp()</i>	<i>cexp()</i>	<i>exp()</i>
<i>log()</i>	<i>clog()</i>	<i>log()</i>
<i>pow()</i>	<i>cpow()</i>	<i>pow()</i>
<i>sqrt()</i>	<i>csqrt()</i>	<i>sqrt()</i>
<i>fabs()</i>	<i>cabs()</i>	<i>fabs()</i>

<tgmath.h>

Headers

13987 If at least one argument for a generic parameter is complex, then use of the macro invokes a
13988 complex function; otherwise, use of the macro invokes a real function.

13989 For each unsuffixed function in the <math.h> header without a *c*-prefixed counterpart in the
13990 XSI <complex.h> header, except for *modf()*, *j0()*, *j1()*, *jn()*, *y0()*, *y1()*, and *yn()*, the corresponding
13991 type-generic macro has the same name as the function. These type-generic macros are:

13992	<i>atan2()</i>	<i>fma()</i>	<i>llround()</i>	<i>remainder()</i>
13993	<i>cbrt()</i>	<i>fmax()</i>	<i>log10()</i>	<i>remquo()</i>
13994	<i>ceil()</i>	<i>fmin()</i>	<i>log1p()</i>	<i>rint()</i>
13995	<i>copysign()</i>	<i>fmod()</i>	<i>log2()</i>	<i>round()</i>
13996	<i>erf()</i>	<i>frexp()</i>	<i>logb()</i>	<i>scalbn()</i>
13997	<i>erfc()</i>	<i>hypot()</i>	<i>lrint()</i>	<i>scalbln()</i>
13998	<i>exp2()</i>	<i>ilogb()</i>	<i>lround()</i>	<i>tgamma()</i>
13999	<i>expm1()</i>	<i>ldexp()</i>	<i>nearbyint()</i>	<i>trunc()</i>
14000	<i>fdim()</i>	<i>lgamma()</i>	<i>nextafter()</i>	
14001	<i>floor()</i>	<i>llrint()</i>	<i>nexttoward()</i>	

14002 If all arguments for generic parameters are real, then use of the macro invokes a real function;
14003 otherwise, use of the macro results in undefined behavior.

14004 For each unsuffixed function in the <complex.h> header that is not a *c*-prefixed counterpart to a
14005 function in the <math.h> header, the corresponding type-generic macro has the same name as
14006 the function. These type-generic macros are:

14007 *carg()*
14008 *cimag()*
14009 *conj()*
14010 *cproj()*
14011 *creal()*

14012 Use of the macro with any real or complex argument invokes a complex function.

APPLICATION USAGE

14013 With the declarations:

```
14014 #include <tgmath.h>
14015 int n;
14016 float f;
14017 double d;
14018 long double ld;
14019 float complex fc;
14020 double complex dc;
14021 long double complex ldc;
```

14022 functions invoked by use of type-generic macros are shown in the following table:

Macro	Use Invokes
<i>exp(n)</i>	<i>exp(n)</i> , the function
<i>acosh(f)</i>	<i>acoshf(f)</i>
<i>sin(d)</i>	<i>sin(d)</i> , the function
<i>atan(ld)</i>	<i>atanl(ld)</i>
<i>log(fc)</i>	<i>clogf(fc)</i>
<i>sqrt(dc)</i>	<i>csqrt(dc)</i>

Macro	Use Invokes
<i>pow(ldc,f)</i>	<i>cpowl(ldc,f)</i>
<i>remainder(n,n)</i>	<i>remainder(n,n)</i> , the function
<i>nextafter(d,f)</i>	<i>nextafter(d,f)</i> , the function
<i>nexttoward(f,ld)</i>	<i>nexttowardf(f,ld)</i>
<i>copysign(n,ld)</i>	<i>copysignl(n,ld)</i>
<i>ceil(fc)</i>	Undefined behavior
<i>rint(dc)</i>	Undefined behavior
<i>fmax(ldc,ld)</i>	Undefined behavior
<i>carg(n)</i>	<i>carg(n)</i> , the function
<i>cproj(f)</i>	<i>cprojf(f)</i>
<i>creal(d)</i>	<i>creal(d)</i> , the function
<i>cimag(ld)</i>	<i>cimagl(ld)</i>
<i>cabs(fc)</i>	<i>cabsf(fc)</i>
<i>carg(dc)</i>	<i>carg(dc)</i> , the function
<i>cproj(ldc)</i>	<i>cprojl(ldc)</i>

RATIONALE

Type-generic macros allow calling a function whose type is determined by the argument type, as is the case for C operators such as '+' and '*'. For example, with a type-generic *cos()* macro, the expression *cos((float)x)* will have type **float**. This feature enables writing more portably efficient code and alleviates need for awkward casting and suffixing in the process of porting or adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

The only arguments that affect the type resolution are the arguments corresponding to the parameters that have type **double** in the synopsis. Hence the type of a type-generic call to *nexttoward()*, whose second parameter is **long double** in the synopsis, is determined solely by the type of the first argument.

The term "type-generic" was chosen over the proposed alternatives of intrinsic and overloading. The term is more specific than intrinsic, which already is widely used with a more general meaning, and reflects a closer match to Fortran's generic functions than to C++ overloading.

The macros are placed in their own header in order not to silently break old programs that include the <math.h> header; for example, with:

```
printf ("%e", sin(x))
```

*modf(double, double *)* is excluded because no way was seen to make it safe without complicating the type resolution.

The implementation might, as an extension, endow appropriate ones of the macros that POSIX.1-2008 specifies only for real arguments with the ability to invoke the complex functions.

POSIX.1-2008 does not prescribe any particular implementation mechanism for generic macros. It could be implemented simply with built-in macros. The generic macro for *sqrt()*, for example, could be implemented with:

```
#undef sqrt
#define sqrt(x) __BUILTIN_GENERIC_sqrt(x)
```

Generic macros are designed for a useful level of consistency with C++ overloaded math functions.

The great majority of existing C programs are expected to be unaffected when the <tgmath.h> header is included instead of the <math.h> or <complex.h> headers. Generic macros are similar to the ISO/IEC 9899:1999 standard library masking macros, though the semantic types of return

<tgmath.h>

Headers

14077 values differ.

14078 The ability to overload on integer as well as floating types would have been useful for some
 14079 functions; for example, *copysign()*. Overloading with different numbers of arguments would
 14080 have allowed reusing names; for example, *remainder()* for *remquo()*. However, these facilities
 14081 would have complicated the specification; and their natural consistent use, such as for a floating
 14082 *abs()* or a two-argument *atan()*, would have introduced further inconsistencies with the
 14083 ISO/IEC 9899:1999 standard for insufficient benefit.

14084 The ISO C standard in no way limits the implementation's options for efficiency, including
 14085 inlining library functions.

14086 FUTURE DIRECTIONS

14087 None.

14088 SEE ALSO

14089 <math.h>, <complex.h>

14090 XSH *cabs()*, *fabs()*, *modf()*

14091 CHANGE HISTORY

14092 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

14093 Issue 7

14094 Austin Group Interpretation 1003.1-2001 #184 is applied, clarifying the functions for which a
 14095 corresponding type-generic macro exists with the same name as the function.

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

Headers

<time.h>

14096 NAME

14097 time.h — time types

14098 SYNOPSIS

14099 #include <time.h>

14100 DESCRIPTION

14101 CX Some of the functionality described on this reference page extends the ISO C standard.
 14102 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to
 14103 enable the visibility of these symbols in this header.

14104 The <time.h> header shall define the **clock_t**, **size_t**, **time_t**, types as described in
 14105 <sys/types.h>.

14106 CX The <time.h> header shall define the **clockid_t** and **timer_t** types as described in <sys/types.h>.

14107 CX The <time.h> header shall define the **locale_t** type as described in <locale.h>.

14108 CPT The <time.h> header shall define the **pid_t** type as described in <sys/types.h>.

14109 CX The tag **sigevent** shall be declared as naming an incomplete structure type, the contents of which
 14110 are described in the <signal.h> header.

14111 The <time.h> header shall declare the **tm** structure, which shall include at least the following
 14112 members:

14113	int	tm_sec	Seconds [0,60].
14114	int	tm_min	Minutes [0,59].
14115	int	tm_hour	Hour [0,23].
14116	int	tm_mday	Day of month [1,31].
14117	int	tm_mon	Month of year [0,11].
14118	int	tm_year	Years since 1900.
14119	int	tm_wday	Day of week [0,6] (Sunday =0).
14120	int	tm_yday	Day of year [0,365].
14121	int	tm_isdst	Daylight Savings flag.

14122 The value of *tm_isdst* shall be positive if Daylight Savings Time is in effect, 0 if Daylight Savings
 14123 Time is not in effect, and negative if the information is not available.

14124 CX The <time.h> header shall declare the **timespec** structure, which shall include at least the
 14125 following members:

14126	time_t	tv_sec	Seconds.
14127	long	tv_nsec	Nanoseconds.

14128 The <time.h> header shall also declare the **itimerspec** structure, which shall include at least the
 14129 following members:

14130	struct timespec	it_interval	Timer period.
14131	struct timespec	it_value	Timer expiration.

14132 The <time.h> header shall define the following macros:

14133 NULL As described in <stddef.h>.

14134 CLOCKS_PER_SEC A number used to convert the value returned by the *clock()* function into
 14135 XSI seconds. The value shall be an expression with type **clock_t**. The value of
 14136 CLOCKS_PER_SEC shall be 1 million on XSI-conformant systems.

<time.h>

Headers

14137		However, it may be variable on other systems, and it should not be
14138		assumed that <code>CLOCKS_PER_SEC</code> is a compile-time constant.
14139	CX	The <time.h> header shall define the following symbolic constants. The values shall have a type
14140		that is assignment-compatible with <code>clockid_t</code> .
14141	MON	<code>CLOCK_MONOTONIC</code>
14142		The identifier for the system-wide monotonic clock, which is defined as a
14143		clock measuring real time, whose value cannot be set via <code>clock_settime()</code>
14144		and which cannot have negative clock jumps. The maximum possible
14145		clock jump shall be implementation-defined.
14146	CPT	<code>CLOCK_PROCESS_CPUTIME_ID</code>
14147		The identifier of the CPU-time clock associated with the process making a
14148		<code>clock()</code> or <code>timer*()</code> function call.
14149	CX	<code>CLOCK_REALTIME</code> The identifier of the system-wide clock measuring real time.
14150	TCT	<code>CLOCK_THREAD_CPUTIME_ID</code>
14151		The identifier of the CPU-time clock associated with the thread making a
14152		<code>clock()</code> or <code>timer*()</code> function call.
14153		The <time.h> header shall define the following symbolic constant:
14154	TIMER_ABSTIME	Flag indicating time is absolute. For functions taking timer objects, this
14155		refers to the clock associated with the timer.
14156	XSI	The <time.h> header shall provide a declaration or definition for <code>getdate_err</code> . The <code>getdate_err</code>
14157		symbol shall expand to an expression of type <code>int</code> . It is unspecified whether <code>getdate_err</code> is a macro
14158		or an identifier declared with external linkage, and whether or not it is a modifiable lvalue. If a
14159		macro definition is suppressed in order to access an actual object, or a program defines an
14160		identifier with the name <code>getdate_err</code> , the behavior is undefined.
14161		The following shall be declared as functions and may also be defined as macros. Function
14162		prototypes shall be provided.
14163	OB	<code>char *asctime(const struct tm *);</code>
14164	OB CX	<code>char *asctime_r(const struct tm *restrict, char *restrict);</code>
14165		<code>clock_t clock(void);</code>
14166	CPT	<code>int clock_getcpuclockid(pid_t, clockid_t *);</code>
14167	CX	<code>int clock_getres(clockid_t, struct timespec *);</code>
14168		<code>int clock_gettime(clockid_t, struct timespec *);</code>
14169		<code>int clock_nanosleep(clockid_t, int, const struct timespec *,</code>
14170		<code>struct timespec *);</code>
14171		<code>int clock_settime(clockid_t, const struct timespec *);</code>
14172	OB	<code>char *ctime(const time_t *);</code>
14173	OB CX	<code>char *ctime_r(const time_t *, char *);</code>
14174		<code>double difftime(time_t, time_t);</code>
14175	XSI	<code>struct tm *getdate(const char *);</code>
14176		<code>struct tm *gmtime(const time_t *);</code>
14177	CX	<code>struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);</code>
14178		<code>struct tm *localtime(const time_t *);</code>
14179	CX	<code>struct tm *localtime_r(const time_t *restrict, struct tm *restrict);</code>
14180		<code>time_t mktime(struct tm *);</code>
14181	CX	<code>int nanosleep(const struct timespec *, struct timespec *);</code>
14182		<code>size_t strftime(char *restrict, size_t, const char *restrict,</code>
14183		<code>const struct tm *restrict);</code>

Headers

<time.h>

```

14184 CX      size_t      strftime_l(char *restrict, size_t, const char *restrict,
14185           const struct tm *restrict, locale_t);
14186 XSI      char        *strptime(const char *restrict, const char *restrict,
14187           struct tm *restrict);
14188           time_t      time(time_t *);
14189 CX      int          timer_create(clockid_t, struct sigevent *restrict,
14190           timer_t *restrict);
14191           int          timer_delete(timer_t);
14192           int          timer_getoverrun(timer_t);
14193           int          timer_gettime(timer_t, struct itimerspec *);
14194           int          timer_settime(timer_t, int, const struct itimerspec *restrict,
14195           struct itimerspec *restrict);
14196           void          tzset(void);

```

14197 The <time.h> header shall declare the following as variables:

```

14198 XSI      extern int      daylight;
14199           extern long     timezone;
14200 CX      extern char      *tzname[];

```

14201 CX Inclusion of the <time.h> header may make visible all symbols from the <signal.h> header.

14202 APPLICATION USAGE

14203 The range [0,60] for *tm_sec* allows for the occasional leap second.

14204 *tm_year* is a signed value; therefore, years before 1900 may be represented.

14205 To obtain the number of clock ticks per second returned by the *times()* function, applications
 14206 should call *sysconf(_SC_CLK_TCK)*.

14207 RATIONALE

14208 The range [0,60] seconds allows for positive or negative leap seconds. The formal definition of
 14209 UTC does not permit double leap seconds, so all mention of double leap seconds has been
 14210 removed, and the range shortened from the former [0,61] seconds seen in earlier versions of this
 14211 standard.

14212 FUTURE DIRECTIONS

14213 None.

14214 SEE ALSO

14215 <locale.h>, <signal.h>, <stddef.h>, <sys/types.h>

14216 XSH Section 2.2 (on page 468), *asctime()*, *clock()*, *clock_getcpuclockid()*, *clock_getres()*,
 14217 *clock_nanosleep()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *localtime()*, *mktime()*, *mq_receive()*,
 14218 *mq_send()*, *nanosleep()*, *pthread_getcpuclockid()*, *pthread_mutex_timedlock()*,
 14219 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *sem_timedwait()*, *strftime()*, *strptime()*,
 14220 *sysconf()*, *time()*, *timer_create()*, *timer_delete()*, *timer_getoverrun()*, *tzset()*, *utime()*

14221 CHANGE HISTORY

14222 First released in Issue 1. Derived from Issue 1 of the SVID.

14223 Issue 5

14224 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 14225 Threads Extension.

<time.h>

Headers

Issue 6

The Open Group Corrigendum U035/6 is applied. In the DESCRIPTION, the types **clockid_t** and **timer_t** have been described.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The POSIX timer-related functions are marked as part of the Timers option.

The symbolic name CLK_TCK is removed. Application usage is added describing how its equivalent functionality can be obtained using *sysconf()*.

The *clock_getcpuclockid()* function and manifest constants CLOCK_PROCESS_CPUTIME_ID and CLOCK_THREAD_CPUTIME_ID are added for alignment with IEEE Std 1003.1d-1999.

The manifest constant CLOCK_MONOTONIC and the *clock_nanosleep()* function are added for alignment with IEEE Std 1003.1j-2000.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The range for seconds is changed from [0,61] to [0,60].
- The **restrict** keyword is added to the prototypes for *asctime_r()*, *gmtime_r()*, *localtime_r()*, *strftime()*, *strptime()*, *timer_create()*, and *timer_settime()*.

IEEE PASC Interpretation 1003.1 #84 is applied adding the statement that symbols from the <signal.h> header may be made visible when the <time.h> header is included.

Extensions beyond the ISO C standard are marked.

Issue 7

Austin Group Interpretation 1003.1-2001 #111 is applied.

SD5-XBD-ERN-74 is applied.

The *strptime_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

Functionality relating to the Timers option is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants, and declarations for the **locale_t** and **pid_t** types and the **sigevent** structure are added.

The description of the *getdate_err* value is expanded.

Headers

<trace.h>

14253 NAME

14254 trace.h — tracing

14255 SYNOPSIS

14256 OB TRC #include <trace.h>

14257 DESCRIPTION

14258 The <trace.h> header shall define the **posix_trace_event_info** structure, which shall include at
 14259 least the following members:

14260	trace_event_id_t	posix_event_id
14261	pid_t	posix_pid
14262	void	*posix_prog_address
14263	pthread_t	posix_thread_id
14264	struct timespec	posix_timestamp
14265	int	posix_truncation_status

14266 The <trace.h> header shall define the **posix_trace_status_info** structure, which shall include at
 14267 least the following members:

14268	int	posix_stream_full_status
14269	int	posix_stream_overrun_status
14270	int	posix_stream_status
14271 OB TRL	int	posix_log_full_status
14272	int	posix_log_overrun_status
14273	int	posix_stream_flush_error
14274	int	posix_stream_flush_status

14275 The <trace.h> header shall define the following symbolic constants:

14276	POSIX_TRACE_ALL_EVENTS
14277 OB TRL	POSIX_TRACE_APPEND
14278 OB TRI	POSIX_TRACE_CLOSE_FOR_CHILD
14279 OB TEF	POSIX_TRACE_FILTER
14280 OB TRL	POSIX_TRACE_FLUSH
14281	POSIX_TRACE_FLUSH_START
14282	POSIX_TRACE_FLUSH_STOP
14283	POSIX_TRACE_FLUSHING
14284	POSIX_TRACE_FULL
14285	POSIX_TRACE_LOOP
14286	POSIX_TRACE_NO_OVERRUN
14287 OB TRL	POSIX_TRACE_NOT_FLUSHING
14288	POSIX_TRACE_NOT_FULL
14289 OB TRI	POSIX_TRACE_INHERITED
14290	POSIX_TRACE_NOT_TRUNCATED
14291	POSIX_TRACE_OVERFLOW
14292	POSIX_TRACE_OVERRUN
14293	POSIX_TRACE_RESUME
14294	POSIX_TRACE_RUNNING
14295	POSIX_TRACE_START
14296	POSIX_TRACE_STOP
14297	POSIX_TRACE_SUSPENDED
14298	POSIX_TRACE_SYSTEM_EVENTS
14299	POSIX_TRACE_TRUNCATED_READ

<trace.h>

Headers

14300 POSIX_TRACE_TRUNCATED_RECORD
 14301 POSIX_TRACE_UNNAMED_USER_EVENT
 14302 POSIX_TRACE_UNTIL_FULL
 14303 POSIX_TRACE_WOPID_EVENTS

14304 OB TEF The <trace.h> header shall define the **size_t**, **trace_attr_t**, **trace_event_id_t**, **trace_event_set_t**,
 14305 and **trace_id_t** types as described in <sys/types.h>.

14306 The following shall be declared as functions and may also be defined as macros. Function
 14307 prototypes shall be provided.

14308 int posix_trace_attr_destroy(trace_attr_t *);
 14309 int posix_trace_attr_getclockres(const trace_attr_t *,
 14310 struct timespec *);
 14311 int posix_trace_attr_getcreatetime(const trace_attr_t *,
 14312 struct timespec *);
 14313 int posix_trace_attr_getgenversion(const trace_attr_t *, char *);
 14314 TRI int posix_trace_attr_getinherited(const trace_attr_t *restrict,
 14315 int *restrict);
 14316 TRL int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict,
 14317 int *restrict);
 14318 int posix_trace_attr_getlogsize(const trace_attr_t *restrict,
 14319 size_t *restrict);
 14320 int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict,
 14321 size_t *restrict);
 14322 int posix_trace_attr_getmaxsystemeventsize(const trace_attr_t *restrict,
 14323 size_t *restrict);
 14324 int posix_trace_attr_getmaxusereventsize(const trace_attr_t *restrict,
 14325 size_t, size_t *restrict);
 14326 int posix_trace_attr_getname(const trace_attr_t *, char *);
 14327 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict,
 14328 int *restrict);
 14329 int posix_trace_attr_getstreamsize(const trace_attr_t *restrict,
 14330 size_t *restrict);
 14331 int posix_trace_attr_init(trace_attr_t *);
 14332 TRI int posix_trace_attr_setinherited(trace_attr_t *, int);
 14333 TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *, int);
 14334 int posix_trace_attr_setlogsize(trace_attr_t *, size_t);
 14335 int posix_trace_attr_setmaxdatasize(trace_attr_t *, size_t);
 14336 int posix_trace_attr_setname(trace_attr_t *, const char *);
 14337 int posix_trace_attr_setstreamfullpolicy(trace_attr_t *, int);
 14338 int posix_trace_attr_setstreamsize(trace_attr_t *, size_t);
 14339 int posix_trace_clear(trace_id_t);
 14340 TRL int posix_trace_close(trace_id_t);
 14341 int posix_trace_create(pid_t, const trace_attr_t *restrict,
 14342 trace_id_t *restrict);
 14343 TRL int posix_trace_create_withlog(pid_t, const trace_attr_t *restrict,
 14344 int, trace_id_t *restrict);
 14345 void posix_trace_event(trace_event_id_t, const void *restrict, size_t);
 14346 int posix_trace_eventid_equal(trace_id_t, trace_event_id_t,
 14347 trace_event_id_t);
 14348 int posix_trace_eventid_get_name(trace_id_t, trace_event_id_t, char *);
 14349 int posix_trace_eventid_open(const char *restrict,

Headers

<trace.h>

```

14350         trace_event_id_t *restrict);
14351 TEF    int  posix_trace_eventset_add(trace_event_id_t, trace_event_set_t *);
14352       int  posix_trace_eventset_del(trace_event_id_t, trace_event_set_t *);
14353       int  posix_trace_eventset_empty(trace_event_set_t *);
14354       int  posix_trace_eventset_fill(trace_event_set_t *, int);
14355       int  posix_trace_eventset_ismember(trace_event_id_t,
14356         const trace_event_set_t *restrict, int *restrict);
14357       int  posix_trace_eventtypelist_getnext_id(trace_id_t,
14358         trace_event_id_t *restrict, int *restrict);
14359       int  posix_trace_eventtypelist_rewind(trace_id_t);
14360 TRL    int  posix_trace_flush(trace_id_t);
14361       int  posix_trace_get_attr(trace_id_t, trace_attr_t *);
14362 TEF    int  posix_trace_get_filter(trace_id_t, trace_event_set_t *);
14363       int  posix_trace_get_status(trace_id_t,
14364         struct posix_trace_status_info *);
14365       int  posix_trace_getnext_event(trace_id_t,
14366         struct posix_trace_event_info *restrict, void *restrict,
14367         size_t, size_t *restrict, int *restrict);
14368 TRL    int  posix_trace_open(int, trace_id_t *);
14369       int  posix_trace_rewind(trace_id_t);
14370 TEF    int  posix_trace_set_filter(trace_id_t, const trace_event_set_t *, int);
14371       int  posix_trace_shutdown(trace_id_t);
14372       int  posix_trace_start(trace_id_t);
14373       int  posix_trace_stop(trace_id_t);
14374       int  posix_trace_timedgetnext_event(trace_id_t,
14375         struct posix_trace_event_info *restrict, void *restrict,
14376         size_t, size_t *restrict, int *restrict,
14377         const struct timespec *restrict);
14378 TEF    int  posix_trace_trid_eventid_open(trace_id_t, const char *restrict,
14379         trace_event_id_t *restrict);
14380       int  posix_trace_trygetnext_event(trace_id_t,
14381         struct posix_trace_event_info *restrict, void *restrict, size_t,
14382         size_t *restrict, int *restrict);

```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The <trace.h> header may be removed in a future version.

SEE ALSO

<sys/types.h>

XSH Section 2.11 (on page 529), *posix_trace_attr_destroy()*, *posix_trace_attr_getclockres()*, *posix_trace_attr_getinherited()*, *posix_trace_attr_getlogsize()*, *posix_trace_clear()*, *posix_trace_close()*, *posix_trace_create()*, *posix_trace_event()*, *posix_trace_eventid_equal()*, *posix_trace_eventset_add()*, *posix_trace_eventtypelist_getnext_id()*, *posix_trace_get_attr()*, *posix_trace_get_filter()*, *posix_trace_getnext_event()*, *posix_trace_start()*

<trace.h>

Headers

14396 **CHANGE HISTORY**

14397 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

14398 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/40 is applied, adding the TRL margin
14399 code to the *posix_trace_flush()* function, for alignment with the System Interfaces volume of
14400 POSIX.1-2008.

14401 **Issue 7**14402 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size_t** type.

14403 The <trace.h> header is marked obsolescent.

14404 This reference page is clarified with respect to macros and symbolic constants.

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

*Headers***<ulimit.h>**14405 **NAME**14406 **ulimit.h** — ulimit commands14407 **SYNOPSIS**14408 OB XSI `#include <ulimit.h>`14409 **DESCRIPTION**14410 The **<ulimit.h>** header shall define the symbolic constants used by the *ulimit()* function.

14411 Symbolic constants:

14412 **UL_GETFSIZE** Get maximum file size.14413 **UL_SETFSIZE** Set maximum file size.14414 The following shall be declared as a function and may also be defined as a macro. A function
14415 prototype shall be provided.14416 `long ulimit(int, ...);`14417 **APPLICATION USAGE**14418 See *ulimit()*.14419 **RATIONALE**

14420 None.

14421 **FUTURE DIRECTIONS**

14422 None.

14423 **SEE ALSO**14424 XSH *ulimit()*14425 **CHANGE HISTORY**

14426 First released in Issue 3.

14427 **Issue 7**14428 The **<ulimit.h>** header is marked obsolescent.

<unistd.h>

Headers

14429 NAME

14430 unistd.h — standard symbolic constants and types

14431 SYNOPSIS

14432 #include <unistd.h>

14433 DESCRIPTION

14434 The <unistd.h> header defines miscellaneous symbolic constants and types, and declares
 14435 miscellaneous functions. The actual values of the constants are unspecified except as shown. The
 14436 contents of this header are shown below.

14437 Version Test Macros

14438 The <unistd.h> header shall define the following symbolic constants. The values shall be
 14439 suitable for use in #if preprocessing directives.

14440 _POSIX_VERSION

14441 Integer value indicating version of this standard (C-language binding) to which the
 14442 implementation conforms. For implementations conforming to POSIX.1-2008, the value
 14443 shall be 200809L.

14444 _POSIX2_VERSION

14445 Integer value indicating version of the Shell and Utilities volume of POSIX.1 to which the
 14446 implementation conforms. For implementations conforming to POSIX.1-2008, the value
 14447 shall be 200809L.

14448 The <unistd.h> header shall define the following symbolic constant only if the implementation
 14449 supports the XSI option; see [Section 2.1.4](#) (on page 19). If defined, its value shall be suitable for
 14450 use in #if preprocessing directives.

14451 XSI _XOPEN_VERSION

14452 Integer value indicating version of the X/Open Portability Guide to which the
 14453 implementation conforms. The value shall be 700.

14454 Constants for Options and Option Groups

14455 The following symbolic constants, if defined in <unistd.h>, shall have a value of -1, 0, or
 14456 greater, unless otherwise specified below. The values shall be suitable for use in #if
 14457 preprocessing directives.

14458 If a symbolic constant is not defined or is defined with the value -1, the option is not supported
 14459 for compilation. If it is defined with a value greater than zero, the option shall always be
 14460 supported when the application is executed. If it is defined with the value zero, the option shall
 14461 be supported for compilation and might or might not be supported at runtime. See [Section 2.1.6](#)
 14462 (on page 26) for further information about the conformance requirements of these three
 14463 categories of support.

14464 ADV _POSIX_ADVISORY_INFO

14465 The implementation supports the Advisory Information option. If this symbol is defined in
 14466 <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
 14467 `sysconf()` shall either be -1 or 200809L.

14468 _POSIX_ASYNCHRONOUS_IO

14469 The implementation supports asynchronous input and output. This symbol shall always be
 14470 set to the value 200809L.

Headers

<unistd.h>

14471		_POSIX_BARRIERS
14472		The implementation supports barriers. This symbol shall always be set to the value
14473		200809L.
14474		_POSIX_CHOWN_RESTRICTED
14475		The use of <i>chown()</i> and <i>fchown()</i> is restricted to a process with appropriate privileges, and
14476		to changing the group ID of a file only to the effective group ID of the process or to one of
14477		its supplementary group IDs. This symbol shall be defined with a value other than -1.
14478		_POSIX_CLOCK_SELECTION
14479		The implementation supports clock selection. This symbol shall always be set to the value
14480		200809L.
14481	CPT	_POSIX_CPUTIME
14482		The implementation supports the Process CPU-Time Clocks option. If this symbol is
14483		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14484		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14485	FSC	_POSIX_FSYNC
14486		The implementation supports the File Synchronization option. If this symbol is defined in
14487		<unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
14488		<i>sysconf()</i> shall either be -1 or 200809L.
14489		_POSIX_IPV6
14490		The implementation supports the IPv6 option. If this symbol is defined in <unistd.h>, it
14491		shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall
14492		either be -1 or 200809L.
14493		_POSIX_JOB_CONTROL
14494		The implementation supports job control. This symbol shall always be set to a value greater
14495		than zero.
14496		_POSIX_MAPPED_FILES
14497		The implementation supports memory mapped Files. This symbol shall always be set to the
14498		value 200809L.
14499	ML	_POSIX_MEMLOCK
14500		The implementation supports the Process Memory Locking option. If this symbol is defined
14501		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14502		by <i>sysconf()</i> shall either be -1 or 200809L.
14503	MLR	_POSIX_MEMLOCK_RANGE
14504		The implementation supports the Range Memory Locking option. If this symbol is defined
14505		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14506		by <i>sysconf()</i> shall either be -1 or 200809L.
14507		_POSIX_MEMORY_PROTECTION
14508		The implementation supports memory protection. This symbol shall always be set to the
14509		value 200809L.
14510	MSG	_POSIX_MESSAGE_PASSING
14511		The implementation supports the Message Passing option. If this symbol is defined in
14512		<unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
14513		<i>sysconf()</i> shall either be -1 or 200809L.
14514	MON	_POSIX_MONOTONIC_CLOCK
14515		The implementation supports the Monotonic Clock option. If this symbol is defined in
14516		<unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by

<unistd.h>

Headers

14517		<code>sysconf()</code> shall either be <code>-1</code> or <code>200809L</code> .
14518		<code>_POSIX_NO_TRUNC</code>
14519		Pathname components longer than <code>{NAME_MAX}</code> generate an error. This symbol shall be
14520		defined with a value other than <code>-1</code> .
14521	PIO	<code>_POSIX_PRIORITIZED_IO</code>
14522		The implementation supports the Prioritized Input and Output option. If this symbol is
14523		defined in <code><unistd.h></code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol
14524		reported by <code>sysconf()</code> shall either be <code>-1</code> or <code>200809L</code> .
14525	PS	<code>_POSIX_PRIORITY_SCHEDULING</code>
14526		The implementation supports the Process Scheduling option. If this symbol is defined in
14527		<code><unistd.h></code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported by
14528		<code>sysconf()</code> shall either be <code>-1</code> or <code>200809L</code> .
14529	RS	<code>_POSIX_RAW_SOCKETS</code>
14530		The implementation supports the Raw Sockets option. If this symbol is defined in
14531		<code><unistd.h></code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported by
14532		<code>sysconf()</code> shall either be <code>-1</code> or <code>200809L</code> .
14533		<code>_POSIX_READER_WRITER_LOCKS</code>
14534		The implementation supports read-write locks. This symbol shall always be set to the value
14535		<code>200809L</code> .
14536		<code>_POSIX_REALTIME_SIGNALS</code>
14537		The implementation supports realtime signals. This symbol shall always be set to the value
14538		<code>200809L</code> .
14539		<code>_POSIX_REGEX</code>
14540		The implementation supports the Regular Expression Handling option. This symbol shall
14541		always be set to a value greater than zero.
14542		<code>_POSIX_SAVED_IDS</code>
14543		Each process has a saved set-user-ID and a saved set-group-ID. This symbol shall always be
14544		set to a value greater than zero.
14545		<code>_POSIX_SEMAPHORES</code>
14546		The implementation supports semaphores. This symbol shall always be set to the value
14547		<code>200809L</code> .
14548	SHM	<code>_POSIX_SHARED_MEMORY_OBJECTS</code>
14549		The implementation supports the Shared Memory Objects option. If this symbol is defined
14550		in <code><unistd.h></code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported
14551		by <code>sysconf()</code> shall either be <code>-1</code> or <code>200809L</code> .
14552		<code>_POSIX_SHELL</code>
14553		The implementation supports the POSIX shell. This symbol shall always be set to a value
14554		greater than zero.
14555	SPN	<code>_POSIX_SPAWN</code>
14556		The implementation supports the Spawn option. If this symbol is defined in <code><unistd.h></code> , it
14557		shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported by <code>sysconf()</code> shall
14558		either be <code>-1</code> or <code>200809L</code> .
14559		<code>_POSIX_SPIN_LOCKS</code>
14560		The implementation supports spin locks. This symbol shall always be set to the value
14561		<code>200809L</code> .

Headers

<unistd.h>

14562	SS	_POSIX_SPARADIC_SERVER
14563		The implementation supports the Process Sporadic Server option. If this symbol is defined
14564		in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14565		by <i>sysconf()</i> shall either be -1 or 200809L.
14566	SIO	_POSIX_SYNCHRONIZED_IO
14567		The implementation supports the Synchronized Input and Output option. If this symbol is
14568		defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14569		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14570	TSA	_POSIX_THREAD_ATTR_STACKADDR
14571		The implementation supports the Thread Stack Address Attribute option. If this symbol is
14572		defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14573		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14574	TSS	_POSIX_THREAD_ATTR_STACKSIZE
14575		The implementation supports the Thread Stack Size Attribute option. If this symbol is
14576		defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14577		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14578	TCT	_POSIX_THREAD_CPU_TIME
14579		The implementation supports the Thread CPU-Time Clocks option. If this symbol is
14580		defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14581		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14582	TPI	_POSIX_THREAD_PRIO_INHERIT
14583		The implementation supports the Non-Robust Mutex Priority Inheritance option. If this
14584		symbol is defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this
14585		symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14586	TPP	_POSIX_THREAD_PRIO_PROTECT
14587		The implementation supports the Non-Robust Mutex Priority Protection option. If this
14588		symbol is defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this
14589		symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14590	TPS	_POSIX_THREAD_PRIORITY_SCHEDULING
14591		The implementation supports the Thread Execution Scheduling option. If this symbol is
14592		defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14593		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14594	TSH	_POSIX_THREAD_PROCESS_SHARED
14595		The implementation supports the Thread Process-Shared Synchronization option. If this
14596		symbol is defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this
14597		symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14598	RPI	_POSIX_THREAD_ROBUST_PRIO_INHERIT
14599		The implementation supports the Robust Mutex Priority Inheritance option. If this symbol
14600		is defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14601		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14602	RPP	_POSIX_THREAD_ROBUST_PRIO_PROTECT
14603		The implementation supports the Robust Mutex Priority Protection option. If this symbol is
14604		defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
14605		reported by <i>sysconf()</i> shall either be -1 or 200809L.

<unistd.h>

Headers

14606		_POSIX_THREAD_SAFE_FUNCTIONS
14607		The implementation supports thread-safe functions. This symbol shall always be set to the
14608		value 200809L.
14609	TSP	_POSIX_THREAD_SPORADIC_SERVER
14610		The implementation supports the Thread Sporadic Server option. If this symbol is defined
14611		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14612		by <i>sysconf()</i> shall either be -1 or 200809L.
14613		_POSIX_THREADS
14614		The implementation supports threads. This symbol shall always be set to the value
14615		200809L.
14616		_POSIX_TIMEOUTS
14617		The implementation supports timeouts. This symbol shall always be set to the value
14618		200809L.
14619		_POSIX_TIMERS
14620		The implementation supports timers. This symbol shall always be set to the value 200809L.
14621	OB TRC	_POSIX_TRACE
14622		The implementation supports the Trace option. If this symbol is defined in <unistd.h>, it
14623		shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall
14624		either be -1 or 200809L.
14625	OB TEF	_POSIX_TRACE_EVENT_FILTER
14626		The implementation supports the Trace Event Filter option. If this symbol is defined in
14627		<unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
14628		<i>sysconf()</i> shall either be -1 or 200809L.
14629	OB TRI	_POSIX_TRACE_INHERIT
14630		The implementation supports the Trace Inherit option. If this symbol is defined in
14631		<unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
14632		<i>sysconf()</i> shall either be -1 or 200809L.
14633	OB TRL	_POSIX_TRACE_LOG
14634		The implementation supports the Trace Log option. If this symbol is defined in <unistd.h>,
14635		it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i>
14636		shall either be -1 or 200809L.
14637	TYM	_POSIX_TYPED_MEMORY_OBJECTS
14638		The implementation supports the Typed Memory Objects option. If this symbol is defined
14639		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14640		by <i>sysconf()</i> shall either be -1 or 200809L.
14641	OB	_POSIX_V6_ILP32_OFF32
14642		The implementation provides a C-language compilation environment with 32-bit int , long ,
14643		pointer , and off_t types.
14644	OB	_POSIX_V6_ILP32_OFFBIG
14645		The implementation provides a C-language compilation environment with 32-bit int , long ,
14646		and pointer types and an off_t type using at least 64 bits.
14647	OB	_POSIX_V6_LP64_OFF64
14648		The implementation provides a C-language compilation environment with 32-bit int and
14649		64-bit long , pointer , and off_t types.

Headers

<unistd.h>

14650	OB	_POSIX_V6_LPBIG_OFFBIG
14651		The implementation provides a C-language compilation environment with an int type
14652		using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14653		_POSIX_V7_ILP32_OFF32
14654		The implementation provides a C-language compilation environment with 32-bit int , long ,
14655		pointer , and off_t types.
14656		_POSIX_V7_ILP32_OFFBIG
14657		The implementation provides a C-language compilation environment with 32-bit int , long ,
14658		and pointer types and an off_t type using at least 64 bits.
14659		_POSIX_V7_LP64_OFF64
14660		The implementation provides a C-language compilation environment with 32-bit int and
14661		64-bit long , pointer , and off_t types.
14662		_POSIX_V7_LPBIG_OFFBIG
14663		The implementation provides a C-language compilation environment with an int type
14664		using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14665		_POSIX2_C_BIND
14666		The implementation supports the C-Language Binding option. This symbol shall always
14667		have the value 200809L.
14668	CD	_POSIX2_C_DEV
14669		The implementation supports the C-Language Development Utilities option. If this symbol
14670		is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14671		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14672		_POSIX2_CHAR_TERM
14673		The implementation supports the Terminal Characteristics option. The value of this symbol
14674		reported by <i>sysconf()</i> shall either be -1 or a value greater than zero.
14675	FD	_POSIX2_FORT_DEV
14676		The implementation supports the FORTRAN Development Utilities option. If this symbol
14677		is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14678		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14679	FR	_POSIX2_FORT_RUN
14680		The implementation supports the FORTRAN Runtime Utilities option. If this symbol
14681		is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14682		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14683		_POSIX2_LOCALEDEF
14684		The implementation supports the creation of locales by the <i>localedef</i> utility. If this symbol is
14685		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14686		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14687	OB BE	_POSIX2_PBS
14688		The implementation supports the Batch Environment Services and Utilities option. If this
14689		symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this
14690		symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14691	OB BE	_POSIX2_PBS_ACCOUNTING
14692		The implementation supports the Batch Accounting option. If this symbol is defined in
14693		<unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
14694		<i>sysconf()</i> shall either be -1 or 200809L.

<unistd.h>

Headers

14695	OB BE	<u>_POSIX2_PBS_CHECKPOINT</u>
14696		The implementation supports the Batch Checkpoint/Restart option. If this symbol is
14697		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14698		reported by <i>sysconf</i> () shall either be -1 or 200809L.
14699	OB BE	<u>_POSIX2_PBS_LOCATE</u>
14700		The implementation supports the Locate Batch Job Request option. If this symbol is defined
14701		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14702		by <i>sysconf</i> () shall either be -1 or 200809L.
14703	OB BE	<u>_POSIX2_PBS_MESSAGE</u>
14704		The implementation supports the Batch Job Message Request option. If this symbol is
14705		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14706		reported by <i>sysconf</i> () shall either be -1 or 200809L.
14707	OB BE	<u>_POSIX2_PBS_TRACK</u>
14708		The implementation supports the Track Batch Job Request option. If this symbol is defined
14709		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14710		by <i>sysconf</i> () shall either be -1 or 200809L.
14711	SD	<u>_POSIX2_SW_DEV</u>
14712		The implementation supports the Software Development Utilities option. If this symbol is
14713		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14714		reported by <i>sysconf</i> () shall either be -1 or 200809L.
14715	UP	<u>_POSIX2_UPE</u>
14716		The implementation supports the User Portability Utilities option. If this symbol is defined
14717		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14718		by <i>sysconf</i> () shall either be -1 or 200809L.
14719	XSI	<u>_XOPEN_CRYPT</u>
14720		The implementation supports the X/Open Encryption Option Group.
14721		<u>_XOPEN_ENH_I18N</u>
14722		The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option
14723		Group. This symbol shall always be set to a value other than -1.
14724		<u>_XOPEN_REALTIME</u>
14725		The implementation supports the X/Open Realtime Option Group.
14726		<u>_XOPEN_REALTIME_THREADS</u>
14727		The implementation supports the X/Open Realtime Threads Option Group.
14728		<u>_XOPEN_SHM</u>
14729		The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This
14730		symbol shall always be set to a value other than -1.
14731	OB XSR	<u>_XOPEN_STREAMS</u>
14732		The implementation supports the XSI STREAMS Option Group.
14733	XSI	<u>_XOPEN_UNIX</u>
14734		The implementation supports the XSI option.
14735	UU	<u>_XOPEN_UUCP</u>
14736		The implementation supports the UUCP Utilities option. If this symbol is defined in
14737		<unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
14738		<i>sysconf</i> () shall be either -1 or 200809L.

Execution-Time Symbolic Constants

If any of the following symbolic constants are not defined in the <unistd.h> header, the value shall vary depending on the file to which it is applied. If defined, they shall have values suitable for use in **#if** preprocessing directives.

If any of the following symbolic constants are defined to have value **-1** in the <unistd.h> header, the implementation shall not provide the option on any file; if any are defined to have a value other than **-1** in the <unistd.h> header, the implementation shall provide the option on all applicable files.

All of the following values, whether defined as symbolic constants in <unistd.h> or not, may be queried with respect to a specific file using the *pathconf()* or *fpathconf()* functions:

_POSIX_ASYNC_IO

Asynchronous input or output operations may be performed for the associated file.

_POSIX_PRIO_IO

Prioritized input or output operations may be performed for the associated file.

_POSIX_SYNC_IO

Synchronized input or output operations may be performed for the associated file.

If the following symbolic constants are defined in the <unistd.h> header, they apply to files and all paths in all file systems on the implementation:

_POSIX_TIMESTAMP_RESOLUTION

The resolution in nanoseconds for all file timestamps.

_POSIX2_SYMLINKS

Symbolic links can be created.

Constants for Functions

The <unistd.h> header shall define **NULL** as described in <stddef.h>.

The <unistd.h> header shall define the following symbolic constants for use with the *access()* function. The values shall be suitable for use in **#if** preprocessing directives.

F_OK Test for existence of file.

R_OK Test for read permission.

W_OK Test for write permission.

X_OK Test for execute (search) permission.

The constants **F_OK**, **R_OK**, **W_OK**, and **X_OK** and the expressions **R_OK | W_OK**, **R_OK | X_OK**, and **R_OK | W_OK | X_OK** shall all have distinct values.

The <unistd.h> header shall define the following symbolic constants for the *confstr()* function:

_CS_PATH

This is the value for the *PATH* environment variable that finds all standard utilities.

_CS_POSIX_V7_ILP32_OFF32_CFLAGS

If *sysconf(_SC_V7_ILP32_OFF32)* returns **-1**, the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the *c99* utility to build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14778 `_CS_POSIX_V7_ILP32_OFF32_LDFLAGS`
 14779 If `sysconf(_SC_V7_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
 14780 Otherwise, this value is the set of final options to be given to the `c99` utility to build an
 14781 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14782 `_CS_POSIX_V7_ILP32_OFF32_LIBS`
 14783 If `sysconf(_SC_V7_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
 14784 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an
 14785 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14786 `_CS_POSIX_V7_ILP32_OFFBIG_CFLAGS`
 14787 If `sysconf(_SC_V7_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14788 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an
 14789 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
 14790 **off_t** type using at least 64 bits.

14791 `_CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS`
 14792 If `sysconf(_SC_V7_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14793 Otherwise, this value is the set of final options to be given to the `c99` utility to build an
 14794 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
 14795 **off_t** type using at least 64 bits.

14796 `_CS_POSIX_V7_ILP32_OFFBIG_LIBS`
 14797 If `sysconf(_SC_V7_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14798 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an
 14799 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
 14800 **off_t** type using at least 64 bits.

14801 `_CS_POSIX_V7_LP64_OFF64_CFLAGS`
 14802 If `sysconf(_SC_V7_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
 14803 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an
 14804 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t**
 14805 types.

14806 `_CS_POSIX_V7_LP64_OFF64_LDFLAGS`
 14807 If `sysconf(_SC_V7_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
 14808 Otherwise, this value is the set of final options to be given to the `c99` utility to build an
 14809 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t**
 14810 types.

14811 `_CS_POSIX_V7_LP64_OFF64_LIBS`
 14812 If `sysconf(_SC_V7_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
 14813 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an
 14814 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t**
 14815 types.

14816 `_CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS`
 14817 If `sysconf(_SC_V7_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14818 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an
 14819 application using a programming model with an **int** type using at least 32 bits and **long**,
 14820 **pointer**, and **off_t** types using at least 64 bits.

14821 `_CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS`
 14822 If `sysconf(_SC_V7_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14823 Otherwise, this value is the set of final options to be given to the `c99` utility to build an
 14824 application using a programming model with an **int** type using at least 32 bits and **long**,

Headers

<unistd.h>

- 14825 **pointer**, and **off_t** types using at least 64 bits.
- 14826 **_CS_POSIX_V7_LPBIG_OFFBIG_LIBS**
- 14827 If *sysconf*(*_SC_V7_LPBIG_OFFBIG*) returns -1, the meaning of this value is unspecified.
- 14828 Otherwise, this value is the set of libraries to be given to the *c99* utility to build an
- 14829 application using a programming model with an **int** type using at least 32 bits and **long**,
- 14830 **pointer**, and **off_t** types using at least 64 bits.
- 14831 **_CS_POSIX_V7_THREADS_CFLAGS**
- 14832 If *sysconf*(*_SC_POSIX_THREADS*) returns -1, the meaning of this value is unspecified.
- 14833 Otherwise, this value is the set of initial options to be given to the *c99* utility to build a
- 14834 multi-threaded application. These flags are in addition to those associated with any of the
- 14835 other *_CS_POSIX_V7_*_CFLAGS* values used to specify particular type size programing
- 14836 environments.
- 14837 **_CS_POSIX_V7_THREADS_LDFLAGS**
- 14838 If *sysconf*(*_SC_POSIX_THREADS*) returns -1, the meaning of this value is unspecified.
- 14839 Otherwise, this value is the set of final options to be given to the *c99* utility to build a multi-
- 14840 threaded application. These flags are in addition to those associated with any of the other
- 14841 *_CS_POSIX_V7_*_LDFLAGS* values used to specify particular type size programing
- 14842 environments.
- 14843 **_CS_POSIX_V7_WIDTH_RESTRICTED_ENVS**
- 14844 This value is a <newline>-separated list of names of programming environments supported
- 14845 by the implementation in which the widths of the **blksize_t**, **cc_t**, **mode_t**, **nfds_t**, **pid_t**,
- 14846 **ptrdiff_t**, **size_t**, **speed_t**, **ssize_t**, **suseconds_t**, **tcflag_t**, **wchar_t**, and **wint_t** types are no
- 14847 greater than the width of type **long**. The format of each name shall be suitable for use with
- 14848 the *getconf* -v option.
- 14849 **_CS_V7_ENV**
- 14850 This is the value that provides the environment variable information (other than that
- 14851 provided by *_CS_PATH*) that is required by the implementation to create a conforming
- 14852 environment, as described in the implementation's conformance documentation.
- 14853 OB The following symbolic constants are reserved for compatibility with Issue 6:
- 14854 **_CS_POSIX_V6_ILP32_OFF32_CFLAGS**
- 14855 **_CS_POSIX_V6_ILP32_OFF32_LDFLAGS**
- 14856 **_CS_POSIX_V6_ILP32_OFF32_LIBS**
- 14857 **_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS**
- 14858 **_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS**
- 14859 **_CS_POSIX_V6_ILP32_OFFBIG_LIBS**
- 14860 **_CS_POSIX_V6_LP64_OFF64_CFLAGS**
- 14861 **_CS_POSIX_V6_LP64_OFF64_LDFLAGS**
- 14862 **_CS_POSIX_V6_LP64_OFF64_LIBS**
- 14863 **_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS**
- 14864 **_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS**
- 14865 **_CS_POSIX_V6_LPBIG_OFFBIG_LIBS**
- 14866 **_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS**
- 14867 **_CS_V6_ENV**
- 14868 The <unistd.h> header shall define **SEEK_CUR**, **SEEK_END**, and **SEEK_SET** as described in
- 14869 <stdio.h>.
- 14870 The <unistd.h> header shall define the following symbolic constants as possible values for the

<unistd.h>

Headers

14871 *function* argument to the *lockf()* function:

14872 F_LOCK Lock a section for exclusive use.
 14873 F_TEST Test section for locks by other processes.
 14874 F_TLOCK Test and lock a section for exclusive use.
 14875 F_ULOCK Unlock locked sections.

14876 The <unistd.h> header shall define the following symbolic constants for *pathconf()*:

14877 _PC_2_SYMLINKS
 14878 _PC_ALLOC_SIZE_MIN
 14879 _PC_ASYNC_IO
 14880 _PC_CHOWN_RESTRICTED
 14881 _PC_FILESIZEBITS
 14882 _PC_LINK_MAX
 14883 _PC_MAX_CANON
 14884 _PC_MAX_INPUT
 14885 _PC_NAME_MAX
 14886 _PC_NO_TRUNC
 14887 _PC_PATH_MAX
 14888 _PC_PIPE_BUF
 14889 _PC_PRIO_IO
 14890 _PC_REC_INCR_XFER_SIZE
 14891 _PC_REC_MAX_XFER_SIZE
 14892 _PC_REC_MIN_XFER_SIZE
 14893 _PC_REC_XFER_ALIGN
 14894 _PC_SYMLINK_MAX
 14895 _PC_SYNC_IO
 14896 _PC_TIMESTAMP_RESOLUTION
 14897 _PC_VDISABLE

14898 The <unistd.h> header shall define the following symbolic constants for *sysconf()*:

14899 _SC_2_C_BIND
 14900 _SC_2_C_DEV
 14901 _SC_2_CHAR_TERM
 14902 _SC_2_FORT_DEV
 14903 _SC_2_FORT_RUN
 14904 _SC_2_LOCALEDEF
 14905 _SC_2_PBS
 14906 _SC_2_PBS_ACCOUNTING
 14907 _SC_2_PBS_CHECKPOINT
 14908 _SC_2_PBS_LOCATE
 14909 _SC_2_PBS_MESSAGE
 14910 _SC_2_PBS_TRACK
 14911 _SC_2_SW_DEV
 14912 _SC_2_UPE
 14913 _SC_2_VERSION
 14914 _SC_ADVISORY_INFO
 14915 _SC_AIO_LISTIO_MAX
 14916 _SC_AIO_MAX
 14917 _SC_AIO_PRIO_DELTA_MAX
 14918 _SC_ARG_MAX

Headers

<unistd.h>

14919 _SC_ASYNCHRONOUS_IO
 14920 _SC_ATEXIT_MAX
 14921 _SC_BARRIERS
 14922 _SC_BC_BASE_MAX
 14923 _SC_BC_DIM_MAX
 14924 _SC_BC_SCALE_MAX
 14925 _SC_BC_STRING_MAX
 14926 _SC_CHILD_MAX
 14927 _SC_CLK_TCK
 14928 _SC_CLOCK_SELECTION
 14929 _SC_COLL_WEIGHTS_MAX
 14930 _SC_CPUTIME
 14931 _SC_DELAYTIMER_MAX
 14932 _SC_EXPR_NEST_MAX
 14933 _SC_FSYNC
 14934 _SC_GETGR_R_SIZE_MAX
 14935 _SC_GETPW_R_SIZE_MAX
 14936 _SC_HOST_NAME_MAX
 14937 _SC_IOV_MAX
 14938 _SC_IPV6
 14939 _SC_JOB_CONTROL
 14940 _SC_LINE_MAX
 14941 _SC_LOGIN_NAME_MAX
 14942 _SC_MAPPED_FILES
 14943 _SC_MEMLOCK
 14944 _SC_MEMLOCK_RANGE
 14945 _SC_MEMORY_PROTECTION
 14946 _SC_MESSAGE_PASSING
 14947 _SC_MONOTONIC_CLOCK
 14948 _SC_MQ_OPEN_MAX
 14949 _SC_MQ_PRIO_MAX
 14950 _SC_NGROUPS_MAX
 14951 _SC_OPEN_MAX
 14952 _SC_PAGE_SIZE
 14953 _SC_PAGESIZE
 14954 _SC_PRIORITIZED_IO
 14955 _SC_PRIORITY_SCHEDULING
 14956 _SC_RAW_SOCKETS
 14957 _SC_RE_DUP_MAX
 14958 _SC_READER_WRITER_LOCKS
 14959 _SC_REALTIME_SIGNALS
 14960 _SC_REGEX
 14961 _SC_RTSIG_MAX
 14962 _SC_SAVED_IDS
 14963 _SC_SEM_NSEMS_MAX
 14964 _SC_SEM_VALUE_MAX
 14965 _SC_SEMAPHORES
 14966 _SC_SHARED_MEMORY_OBJECTS
 14967 _SC_SHELL
 14968 _SC_SIGQUEUE_MAX
 14969 _SC_SPAWN
 14970 _SC_SPIN_LOCKS

14971 _SC_SPORADIC_SERVER
 14972 _SC_SS_REPL_MAX
 14973 _SC_STREAM_MAX
 14974 _SC_SYMLINK_MAX
 14975 _SC_SYNCHRONIZED_IO
 14976 _SC_THREAD_ATTR_STACKADDR
 14977 _SC_THREAD_ATTR_STACKSIZE
 14978 _SC_THREAD_CPU_TIME
 14979 _SC_THREAD_DESTRUCTOR_ITERATIONS
 14980 _SC_THREAD_KEYS_MAX
 14981 _SC_THREAD_PRIO_INHERIT
 14982 _SC_THREAD_PRIO_PROTECT
 14983 _SC_THREAD_PRIORITY_SCHEDULING
 14984 _SC_THREAD_PROCESS_SHARED
 14985 _SC_THREAD_ROBUST_PRIO_INHERIT
 14986 _SC_THREAD_ROBUST_PRIO_PROTECT
 14987 _SC_THREAD_SAFE_FUNCTIONS
 14988 _SC_THREAD_SPORADIC_SERVER
 14989 _SC_THREAD_STACK_MIN
 14990 _SC_THREAD_THREADS_MAX
 14991 _SC_THREADS
 14992 _SC_TIMEOUTS
 14993 _SC_TIMER_MAX
 14994 _SC_TIMERS
 14995 _SC_TRACE
 14996 _SC_TRACE_EVENT_FILTER
 14997 _SC_TRACE_EVENT_NAME_MAX
 14998 _SC_TRACE_INHERIT
 14999 _SC_TRACE_LOG
 15000 _SC_TRACE_NAME_MAX
 15001 _SC_TRACE_SYS_MAX
 15002 _SC_TRACE_USER_EVENT_MAX
 15003 _SC_TTY_NAME_MAX
 15004 _SC_TYPED_MEMORY_OBJECTS
 15005 _SC_TZNAME_MAX
 15006 _SC_V7_ILP32_OFF32
 15007 _SC_V7_ILP32_OFFBIG
 15008 _SC_V7_LP64_OFF64
 15009 _SC_V7_LPBIG_OFFBIG
 15010 OB _SC_V6_ILP32_OFF32
 15011 _SC_V6_ILP32_OFFBIG
 15012 _SC_V6_LP64_OFF64
 15013 _SC_V6_LPBIG_OFFBIG
 15014 _SC_VERSION
 15015 _SC_XOPEN_CRYPT
 15016 _SC_XOPEN_ENH_I18N
 15017 _SC_XOPEN_REALTIME
 15018 _SC_XOPEN_REALTIME_THREADS
 15019 _SC_XOPEN_SHM
 15020 _SC_XOPEN_STREAMS
 15021 _SC_XOPEN_UNIX
 15022 _SC_XOPEN_UUCP

Headers

<unistd.h>

15023 `_SC_XOPEN_VERSION`

15024 The two constants `_SC_PAGESIZE` and `_SC_PAGE_SIZE` may be defined to have the same value.

15025 The <unistd.h> header shall define the following symbolic constants for file streams:

15026 `STDERR_FILENO` File number of *stderr*; 2.

15027 `STDIN_FILENO` File number of *stdin*; 0.

15028 `STDOUT_FILENO` File number of *stdout*; 1.

15029 The <unistd.h> header shall define the following symbolic constant for terminal special
15030 character handling:

15031 `_POSIX_VDISABLE` This symbol shall be defined to be the value of a character that shall
15032 disable terminal special character handling as described in Section 11.2.6
15033 (on page 212). This symbol shall always be set to a value other than -1.

15034 Type Definitions

15035 The <unistd.h> header shall define the `size_t`, `ssize_t`, `uid_t`, `gid_t`, `off_t`, and `pid_t` types as
15036 described in <sys/types.h>.

15037 The <unistd.h> header shall define the `intptr_t` type as described in <inttypes.h>.

15038 Declarations

15039 The following shall be declared as functions and may also be defined as macros. Function
15040 prototypes shall be provided.

15041	<code>int</code>	<code>access(const char *, int);</code>
15042	<code>unsigned</code>	<code>alarm(unsigned);</code>
15043	<code>int</code>	<code>chdir(const char *);</code>
15044	<code>int</code>	<code>chown(const char *, uid_t, gid_t);</code>
15045	<code>int</code>	<code>close(int);</code>
15046	<code>size_t</code>	<code>confstr(int, char *, size_t);</code>
15047 XSI	<code>char</code>	<code>*crypt(const char *, const char *);</code>
15048 CX	<code>char</code>	<code>*ctermid(char *);</code>
15049	<code>int</code>	<code>dup(int);</code>
15050	<code>int</code>	<code>dup2(int, int);</code>
15051	<code>void</code>	<code>_exit(int);</code>
15052 XSI	<code>void</code>	<code>encrypt(char [64], int);</code>
15053	<code>int</code>	<code>execl(const char *, const char *, ...);</code>
15054	<code>int</code>	<code>execle(const char *, const char *, ...);</code>
15055	<code>int</code>	<code>execlp(const char *, const char *, ...);</code>
15056	<code>int</code>	<code>execv(const char *, char *const []);</code>
15057	<code>int</code>	<code>execve(const char *, char *const [], char *const []);</code>
15058	<code>int</code>	<code>execvp(const char *, char *const []);</code>
15059	<code>int</code>	<code>faccessat(int, const char *, int, int);</code>
15060	<code>int</code>	<code>fchdir(int);</code>
15061	<code>int</code>	<code>fchown(int, uid_t, gid_t);</code>
15062	<code>int</code>	<code>fchownat(int, const char *, uid_t, gid_t, int);</code>
15063 SIO	<code>int</code>	<code>fdatasync(int);</code>
15064	<code>int</code>	<code>fexecve(int, char *const [], char *const []);</code>
15065	<code>pid_t</code>	<code>fork(void);</code>
15066	<code>long</code>	<code>fpathconf(int, int);</code>

<unistd.h>

Headers

15067	FSC	int	fsync(int);
15068		int	ftruncate(int, off_t);
15069		char	*getcwd(char *, size_t);
15070		gid_t	getegid(void);
15071		uid_t	geteuid(void);
15072		gid_t	getgid(void);
15073		int	getgroups(int, gid_t []);
15074	XSI	long	gethostid(void);
15075		int	gethostname(char *, size_t);
15076		char	*getlogin(void);
15077		int	getlogin_r(char *, size_t);
15078		int	getopt(int, char * const [], const char *);
15079		pid_t	getpgid(pid_t);
15080		pid_t	getpgrp(void);
15081		pid_t	getpid(void);
15082		pid_t	getppid(void);
15083		pid_t	getsid(pid_t);
15084		uid_t	getuid(void);
15085		int	isatty(int);
15086		int	lchown(const char *, uid_t, gid_t);
15087		int	link(const char *, const char *);
15088		int	linkat(int, const char *, int, const char *, int);
15089	XSI	int	lockf(int, int, off_t);
15090		off_t	lseek(int, off_t, int);
15091	XSI	int	nice(int);
15092		long	pathconf(const char *, int);
15093		int	pause(void);
15094		int	pipe(int [2]);
15095		ssize_t	pread(int, void *, size_t, off_t);
15096		ssize_t	pwrite(int, const void *, size_t, off_t);
15097		ssize_t	read(int, void *, size_t);
15098		ssize_t	readlink(const char *restrict, char *restrict, size_t);
15099		ssize_t	readlinkat(int, const char *restrict, char *restrict, size_t);
15100		int	rmdir(const char *);
15101		int	setegid(gid_t);
15102		int	seteuid(uid_t);
15103		int	setgid(gid_t);
15104		int	setpgid(pid_t, pid_t);
15105	OB XSI	pid_t	setpgrp(void);
15106	XSI	int	setregid(gid_t, gid_t);
15107		int	setreuid(uid_t, uid_t);
15108		pid_t	setsid(void);
15109		int	setuid(uid_t);
15110		unsigned	sleep(unsigned);
15111	XSI	void	swab(const void *restrict, void *restrict, ssize_t);
15112		int	symlink(const char *, const char *);
15113		int	symlinkat(const char *, int, const char *);
15114	XSI	void	sync(void);
15115		long	sysconf(int);
15116		pid_t	tcgetpgrp(int);
15117		int	tcsetpgrp(int, pid_t);
15118		int	truncate(const char *, off_t);

Headers

<unistd.h>

```

15119 char          *ttyname(int);
15120 int           ttyname_r(int, char *, size_t);
15121 int           unlink(const char *);
15122 int           unlinkat(int, const char *, int);
15123 ssize_t       write(int, const void *, size_t);

```

15124 Implementations may also include the *pthread_atfork()* prototype as defined in <pthread.h>.

15125 The <unistd.h> header shall declare the following external variables:

```

15126 extern char   *optarg;
15127 extern int    opterr, optind, optopt;

```

15128 APPLICATION USAGE

15129 POSIX.1-2008 only describes the behavior of systems that claim conformance to it. However,
 15130 application developers who want to write applications that adapt to other versions of this
 15131 standard (or to systems that do not conform to any POSIX standard) may find it useful to code
 15132 them so as to conditionally compile different code depending on the value of
 15133 `_POSIX_VERSION`, for example:

```

15134 #if _POSIX_VERSION >= 200112L
15135 /* Use the newer function that copes with large files. */
15136 off_t pos=ftello(fp);
15137 #else
15138 /* Either this is an old version of POSIX, or _POSIX_VERSION is
15139    not even defined, so use the traditional function. */
15140 long pos=ftell(fp);
15141 #endif

```

15142 Earlier versions of POSIX.1-2008 and of the Single UNIX Specification can be identified by the
 15143 following macros:

```

15144 POSIX.1-1988 standard
15145     _POSIX_VERSION == 198808L

```

```

15146 POSIX.1-1990 standard
15147     _POSIX_VERSION == 199009L

```

```

15148 ISO POSIX-1: 1996 standard
15149     _POSIX_VERSION == 199506L

```

```

15150 Single UNIX Specification, Version 1
15151     _XOPEN_UNIX and _XOPEN_VERSION == 4

```

```

15152 Single UNIX Specification, Version 2
15153     _XOPEN_UNIX and _XOPEN_VERSION == 500

```

```

15154 ISO POSIX-1: 2001 and Single UNIX Specification, Version 3
15155     _POSIX_VERSION == 200112L, plus (if the XSI option is supported) _XOPEN_UNIX and
15156     _XOPEN_VERSION == 600

```

15157 POSIX.1-2008 does not make any attempt to define application binary interaction with the
 15158 underlying operating system. However, application developers may find it useful to query
 15159 `_SC_VERSION` at runtime via *sysconf()* to determine whether the current version of the
 15160 operating system supports the necessary functionality as in the following program fragment:

```

15161 if (sysconf(_SC_VERSION) < 200809L) {
15162     fprintf(stderr, "POSIX.1-2008 system required, terminating \n");
15163     exit(1);

```

15164 }

15165 New applications should not use `_XOPEN_SHM` or `_XOPEN_ENH_I18N`.

15166 RATIONALE

15167 As POSIX.1-2008 evolved, certain options became sufficiently standardized that it was
15168 concluded that simply requiring one of the option choices was simpler than retaining the option.
15169 However, for backwards-compatibility, the option flags (with required constant values) are
15170 retained.

15171 Version Test Macros

15172 The standard developers considered altering the definition of `_POSIX_VERSION` and removing
15173 `_SC_VERSION` from the specification of `sysconf()` since the utility to an application was deemed
15174 by some to be minimal, and since the implementation of the functionality is potentially
15175 problematic. However, they recognized that support for existing application binaries is a
15176 concern to manufacturers, application developers, and the users of implementations conforming
15177 to POSIX.1-2008.

15178 While the example using `_SC_VERSION` in the APPLICATION USAGE section does not provide
15179 the greatest degree of imaginable utility to the application developer or user, it is arguably better
15180 than a **core** file or some other equally obscure result. (It is also possible for implementations to
15181 encode and recognize application binaries compiled in various POSIX.1-conforming
15182 environments, and modify the semantics of the underlying system to conform to the
15183 expectations of the application.) For the reasons outlined in the preceding paragraphs and in the
15184 APPLICATION USAGE section, the standard developers elected to retain the `_POSIX_VERSION`
15185 and `_SC_VERSION` functionality.

15186 Compile-Time Symbolic Constants for System-Wide Options

15187 POSIX.1-2008 includes support in certain areas for the newly adopted policy governing options
15188 and stubs.

15189 This policy provides flexibility for implementations in how they support options. It also
15190 specifies how conforming applications can adapt to different implementations that support
15191 different sets of options. It allows the following:

- 15192 1. If an implementation has no interest in supporting an option, it does not have to provide
15193 anything associated with that option beyond the announcement that it does not support
15194 it.
- 15195 2. An implementation can support a partial or incompatible version of an option (as a non-
15196 standard extension) as long as it does not claim to support the option.
- 15197 3. An application can determine whether the option is supported. A strictly conforming
15198 application must check this announcement mechanism before first using anything
15199 associated with the option.

15200 There is an important implication of this policy. POSIX.1-2008 cannot dictate the behavior of
15201 interfaces associated with an option when the implementation does not claim to support the
15202 option. In particular, it cannot require that a function associated with an unsupported option
15203 will fail if it does not perform as specified. However, this policy does not prevent a standard
15204 from requiring certain functions to always be present, but that they shall always fail on some
15205 implementations. The `setpgid()` function in the POSIX.1-1990 standard, for example, is
15206 considered appropriate.

15207 The POSIX standards include various options, and the C-language binding support for an

Headers

<unistd.h>

option implies that the implementation must supply data types and function interfaces. An application must be able to discover whether the implementation supports each option.

Any application must consider the following three cases for each option:

1. Option never supported.

The implementation advertises at compile time that the option will never be supported. In this case, it is not necessary for the implementation to supply any of the data types or function interfaces that are provided only as part of the option. The implementation might provide data types and functions that are similar to those defined by POSIX.1-2008, but there is no guarantee for any particular behavior.

2. Option always supported.

The implementation advertises at compile time that the option will always be supported. In this case, all data types and function interfaces shall be available and shall operate as specified.

3. Option might or might not be supported.

Some implementations might not provide a mechanism to specify support of options at compile time. In addition, the implementation might be unable or unwilling to specify support or non-support at compile time. In either case, any application that might use the option at runtime must be able to compile and execute. The implementation must provide, at compile time, all data types and function interfaces that are necessary to allow this. In this situation, there must be a mechanism that allows the application to query, at runtime, whether the option is supported. If the application attempts to use the option when it is not supported, the result is unspecified unless explicitly specified otherwise in POSIX.1-2008.

FUTURE DIRECTIONS

None.

SEE ALSO

<inttypes.h>, <limits.h>, <stddef.h>, <stdio.h>, <sys/socket.h>, <sys/types.h>, <termios.h>, <wctype.h>

XSH *access()*, *alarm()*, *chown()*, *close()*, *confstr()*, *crypt()*, *ctermid()*, *dup()*, *_Exit()*, *encrypt()*, *exec*, *fchdir()*, *fchown()*, *fdatasync()*, *fork()*, *fpathconf()*, *fsync()*, *ftruncate()*, *getcwd()*, *getegid()*, *geteuid()*, *getgid()*, *getgroups()*, *gethostid()*, *gethostname()*, *getlogin()*, *getopt()*, *getpgid()*, *getpgrp()*, *getpid()*, *getppid()*, *getsid()*, *getuid()*, *isatty()*, *lchown()*, *link()*, *lockf()*, *lseek()*, *nice()*, *pause()*, *pipe()*, *read()*, *readlink()*, *rmdir()*, *setegid()*, *seteuid()*, *setgid()*, *setpgid()*, *setpgrp()*, *setregid()*, *setreuid()*, *setsid()*, *setuid()*, *sleep()*, *swab()*, *symlink()*, *sync()*, *sysconf()*, *tcgetpgrp()*, *tcsetpgrp()*, *truncate()*, *ttyname()*, *unlink()*, *write()*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

The symbolic constants `_XOPEN_REALTIME` and `_XOPEN_REALTIME_THREADS` are added. `_POSIX2_C_BIND`, `_XOPEN_ENH_I18N`, and `_XOPEN_SHM` must now be set to a value other than `-1` by a conforming implementation.

Large File System extensions are added.

- 15252 The type of the argument to *sbrk()* is changed from **int** to **intptr_t**.
- 15253 *_XBS_* constants are added to the list of constants for Options and Option Groups, to the list of
 15254 constants for the *confstr()* function, and to the list of constants to the *sysconf()* function. These
 15255 are all marked EX.
- 15256 **Issue 6**
- 15257 *_POSIX2_C_VERSION* is removed.
- 15258 The Open Group Corrigendum U026/4 is applied, adding the prototype for *fdatasync()*.
- 15259 The Open Group Corrigendum U026/1 is applied, adding the symbols *_SC_XOPEN_LEGACY*,
 15260 *_SC_XOPEN_REALTIME*, and *_SC_XOPEN_REALTIME_THREADS*.
- 15261 The symbols *_XOPEN_STREAMS* and *_SC_XOPEN_STREAMS* are added to support the XSI
 15262 STREAMS Option Group.
- 15263 Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in
 15264 IEEE Std 1003.1-2001.
- 15265 The LEGACY symbol *_SC_PASS_MAX* is removed.
- 15266 The following new requirements on POSIX implementations derive from alignment with the
 15267 Single UNIX Specification:
- 15268 • The *_CS_POSIX_** and *_CS_XBS5_** constants are added for the *confstr()* function.
 - 15269 • The *_SC_XBS5_** constants are added for the *sysconf()* function.
 - 15270 • The symbolic constants *F_ULOCK*, *F_LOCK*, *F_TLOCK*, and *F_TEST* are added.
 - 15271 • The **uid_t**, **gid_t**, **off_t**, **pid_t**, and **useconds_t** types are mandated.
- 15272 The *gethostname()* prototype is added for sockets.
- 15273 A new section is added for System-Wide Options.
- 15274 Function prototypes for *setegid()* and *seteuid()* are added.
- 15275 Option symbolic constants are added for *_POSIX_ADVISORY_INFO*, *_POSIX_CPUTIME*,
 15276 *_POSIX_SPAWN*, *_POSIX_SPORADIC_SERVER*, *_POSIX_THREAD_CPUTIME*,
 15277 *_POSIX_THREAD_SPORADIC_SERVER*, and *_POSIX_TIMEOUTS*, and *pathconf()* variables are
 15278 added for *_PC_ALLOC_SIZE_MIN*, *_PC_REC_INCR_XFER_SIZE*, *_PC_REC_MAX_XFER_SIZE*,
 15279 *_PC_REC_MIN_XFER_SIZE*, and *_PC_REC_XFER_ALIGN* for alignment with IEEE Std
 15280 1003.1d-1999.
- 15281 The following are added for alignment with IEEE Std 1003.1j-2000:
- 15282 • Option symbolic constants *_POSIX_BARRIERS*, *_POSIX_CLOCK_SELECTION*,
 15283 *_POSIX_MONOTONIC_CLOCK*, *_POSIX_READER_WRITER_LOCKS*,
 15284 *_POSIX_SPIN_LOCKS*, and *_POSIX_TYPED_MEMORY_OBJECTS*
 - 15285 • *sysconf()* variables *_SC_BARRIERS*, *_SC_CLOCK_SELECTION*,
 15286 *_SC_MONOTONIC_CLOCK*, *_SC_READER_WRITER_LOCKS*, *_SC_SPIN_LOCKS*, and
 15287 *_SC_TYPED_MEMORY_OBJECTS*
- 15288 The *_SC_XBS5* macros associated with the ISO/IEC 9899:1990 standard are marked LEGACY,
 15289 and new equivalent *_SC_V6* macros associated with the ISO/IEC 9899:1999 standard are
 15290 introduced.
- 15291 The *getwd()* function is marked LEGACY.
- 15292 The **restrict** keyword is added to the prototypes for *readlink()* and *swab()*.

Constants for options are now harmonized, so when supported they take the year of approval of IEEE Std 1003.1-2001 as the value.

The following are added for alignment with IEEE Std 1003.1q-2000:

- Optional symbolic constants `_POSIX_TRACE`, `_POSIX_TRACE_EVENT_FILTER`, `_POSIX_TRACE_LOG`, and `_POSIX_TRACE_INHERIT`
- The `sysconf()` symbolic constants `_SC_TRACE`, `_SC_TRACE_EVENT_FILTER`, `_SC_TRACE_LOG`, and `_SC_TRACE_INHERIT`

The `brk()` and `sbrk()` LEGACY functions are removed.

The Open Group Base Resolution bwg2001-006 is applied, which reworks the XSI versioning information.

The Open Group Base Resolution bwg2001-008 is applied, changing the `namelen` parameter for `gethostname()` from `socklen_t` to `size_t`.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/2 is applied, changing “Thread Stack Address Size” to “Thread Stack Size Attribute”.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/20 is applied, adding the `_POSIX_IPV6`, `_SC_V6`, and `_SC_RAW_SOCKETS` symbols.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/21 is applied, correcting the description in “Constants for Functions” for the `_CS_POSIX_V6_LP64_OFF64_CFLAGS`, `_CS_POSIX_V6_LP64_OFF64_LDFLAGS`, and `_CS_POSIX_V6_LP64_OFF64_LIBS` symbols.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/22 is applied, removing the shading for the `_PC*` and `_SC*` constants, since these are mandatory on all implementations.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/23 is applied, adding the `_PC_SYMLINK_MAX` and `_SC_SYMLINK_MAX` constants.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/24 is applied, correcting the shading and margin code for the `fsync()` function.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/25 is applied, adding the following text to the APPLICATION USAGE section: “New applications should not use `_XOPEN_SHM` or `_XOPEN_ENH_I18N`”.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/29 is applied, clarifying the requirements for when constants for Options and Option Groups can be defined or undefined.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/30 is applied, changing the `_V6_ILP32_OFF32`, `_V6_ILP32_OFFBIG`, `_V6_LP64_OFF64`, and `_V6_LP64_OFFBIG` symbols to `_POSIX_V6_ILP32_OFF32`, `_POSIX_V6_ILP32_OFFBIG`, `_POSIX_V6_LP64_OFF64`, and `_POSIX_V6_LP64_OFFBIG`, respectively. This is for consistency with the `sysconf()` and `c99` reference pages.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/31 is applied, adding that the format of names of programming environments can be obtained using the `getconf -v` option.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/32 is applied, deleting the `_SC_FILE_LOCKING`, `_SC_2_C_VERSION`, and `_SC_XOPEN_XCU_VERSION` constants.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/33 is applied, adding `_SC_SS_REPL_MAX`, `_SC_TRACE_EVENT_NAME_MAX`, `_SC_TRACE_NAME_MAX`, `_SC_TRACE_SYS_MAX`, and `_SC_TRACE_USER_EVENT_MAX` to the list of symbolic constants for `sysconf()`.

- 15336 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/34 is applied, updating the prototype for
15337 the *symlink()* function to match that in the System Interfaces volume of IEEE Std 1003.1-2001.
- 15338 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/35 is applied, adding *_PC_2_SYMLINKS*
15339 to the symbolic constants list for *pathconf()*. This corresponds to the definition of
15340 *POSIX2_SYMLINKS* in the Shell and Utilities volume of IEEE Std 1003.1-2001.
- 15341 **Issue 7**
- 15342 Austin Group Interpretations 1003.1-2001 #026 and #047 are applied.
- 15343 Austin Group Interpretation 1003.1-2001 #166 is applied to permit an additional compiler flag to
15344 enable threads.
- 15345 Austin Group Interpretation 1003.1-2001 #178 is applied, clarifying the values allowed for
15346 *_POSIX2_CHAR_TERM*.
- 15347 SD5-XBD-ERN-41 is applied, adding the *_POSIX2_SYMLINKS* constant.
- 15348 SD5-XBD-ERN-76 and SD5-XBD-ERN-77 are applied.
- 15349 Symbols to support the UUCP Utilities option are added.
- 15350 The variables for the supported programming environments are updated to be V7.
- 15351 The LEGACY and obsolescent symbols are removed.
- 15352 The *faccessat()*, *fchownat()*, *fexecve()*, *linkat()*, *readlinkat()*, *symlinkat()*, and *unlinkat()* functions
15353 are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.
- 15354 The *_POSIX_TRACE** constants from the Trace option are marked obsolescent.
- 15355 The *_POSIX2_PBS** constants from the Batch Environment Services and Utilities option are
15356 marked obsolescent.
- 15357 Functionality relating to the Asynchronous Input and Output, Barriers, Clock Selection, Memory
15358 Mapped Files, Memory Protection, Realtime Signals Extension, Semaphores, Spin Locks,
15359 Threads, Timeouts, and Timers options is moved to the Base.
- 15360 Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options
15361 is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex
15362 or Robust Mutex Priority Inheritance, respectively.
- 15363 This reference page is clarified with respect to macros and symbolic constants.
- 15364 Changes are made related to support for finegrained timestamps and the
15365 *_POSIX_TIMESTAMP_RESOLUTION* constant is added.
- 15366 The *_SC_THREAD_ROBUST_PRIO_INHERIT* and *_SC_THREAD_ROBUST_PRIO_PROTECT*
15367 symbolic constants are added.

Headers

<utime.h>

NAME

utime.h — access and modification times structure

SYNOPSIS

```
#include <utime.h>
```

DESCRIPTION

The <utime.h> header shall declare the **utimbuf** structure, which shall include the following members:

time_t	actime	Access time.
time_t	modtime	Modification time.

The times shall be measured in seconds since the Epoch.

The <utime.h> header shall define the **time_t** type as described in <sys/types.h>.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
int utime(const char *, const struct utimbuf *);
```

APPLICATION USAGE

The *utime()* function only allows setting file timestamps to the nearest second. Applications should use the *utimensat()* function instead. See <sys/stat.h>.

RATIONALE

None.

FUTURE DIRECTIONS

The <utime.h> header may be removed in a future version.

SEE ALSO

<sys/stat.h>, <sys/types.h>

XSH *futimens()*, *utime()*

CHANGE HISTORY

First released in Issue 3.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **time_t** type is defined.

Issue 7

The <utime.h> header is marked obsolescent.

<utmpx.h>*Headers*15400 **NAME**

15401 utmpx.h — user accounting database definitions

15402 **SYNOPSIS**

15403 XSI #include <utmpx.h>

15404 **DESCRIPTION**15405 The **<utmpx.h>** header shall define the **utmpx** structure that shall include at least the following
15406 members:

15407	char	ut_user[]	User login name.
15408	char	ut_id[]	Unspecified initialization process identifier.
15409	char	ut_line[]	Device name.
15410	pid_t	ut_pid	Process ID.
15411	short	ut_type	Type of entry.
15412	struct timeval	ut_tv	Time entry was made.

15413 The **<utmpx.h>** header shall define the **pid_t** type through **typedef**, as described in
15414 **<sys/types.h>**.15415 The **<utmpx.h>** header shall define the **timeval** structure as described in **<sys/time.h>**.15416 Inclusion of the **<utmpx.h>** header may also make visible all symbols from **<sys/time.h>**.15417 The **<utmpx.h>** header shall define the following symbolic constants as possible values for the
15418 **ut_type** member of the **utmpx** structure:

15419	EMPTY	No valid user accounting information.
15420	BOOT_TIME	Identifies time of system boot.
15421	OLD_TIME	Identifies time when system clock changed.
15422	NEW_TIME	Identifies time after system clock changed.
15423	USER_PROCESS	Identifies a process.
15424	INIT_PROCESS	Identifies a process spawned by the init process.
15425	LOGIN_PROCESS	Identifies the session leader of a logged-in user.
15426	DEAD_PROCESS	Identifies a session leader who has exited.

15427 The following shall be declared as functions and may also be defined as macros. Function
15428 prototypes shall be provided.

15429	void	endutxent(void);
15430	struct utmpx	*getutxent(void);
15431	struct utmpx	*getutxid(const struct utmpx *);
15432	struct utmpx	*getutxline(const struct utmpx *);
15433	struct utmpx	*pututxline(const struct utmpx *);
15434	void	setutxent(void);

*Headers***<utmpx.h>**15435 **APPLICATION USAGE**

15436 None.

15437 **RATIONALE**

15438 None.

15439 **FUTURE DIRECTIONS**

15440 None.

15441 **SEE ALSO**15442 [<sys/time.h>](#), [<sys/types.h>](#)15443 XSH *endutxent()*15444 **CHANGE HISTORY**

15445 First released in Issue 4, Version 2.

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

<wchar.h>*Headers*15446 **NAME**15447 **wchar.h** — wide-character handling15448 **SYNOPSIS**

15449 #include <wchar.h>

15450 **DESCRIPTION**

15451 CX Some of the functionality described on this reference page extends the ISO C standard.
 15452 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to
 15453 enable the visibility of these symbols in this header.

15454 The <wchar.h> header shall define the following types:

15455 CX **FILE** As described in <stdio.h>.15456 CX **locale_t** As described in <locale.h>.

15457 **mbstate_t** An object type other than an array type that can hold the conversion state
 15458 information necessary to convert between sequences of (possibly multi-byte)
 15459 CX characters and wide characters. If a codeset is being used such that an
 15460 **mbstate_t** needs to preserve more than two levels of reserved state, the results
 15461 are unspecified.

15462 **size_t** As described in <stddef.h>.15463 CX **va_list** As described in <stdarg.h>.15464 **wchar_t** As described in <stddef.h>.

15465 OB XSI **wctype_t** A scalar type of a data object that can hold values which represent locale-
 15466 specific character classification.

15467 **wint_t** An integer type capable of storing any valid value of **wchar_t** or WEOF.

15468 The tag **tm** shall be declared as naming an incomplete structure type, the contents of which are
 15469 described in the <time.h> header.

15470 The implementation shall support one or more programming environments in which the width
 15471 of **wint_t** is no greater than the width of type **long**. The names of these programming
 15472 environments can be obtained using the *confstr()* function or the *getconf* utility.

15473 The <wchar.h> header shall define the following macros:

15474 **WCHAR_MAX** As described in <stdint.h>.15475 **WCHAR_MIN** As described in <stdint.h>.

15476 **WEOF** Constant expression of type **wint_t** that is returned by several WP functions to
 15477 indicate end-of-file.

15478 **NULL** As described in <stddef.h>.

15479 CX Inclusion of the <wchar.h> header may make visible all symbols from the headers <ctype.h>,
 15480 <string.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, and <time.h>.

15481 The following shall be declared as functions and may also be defined as macros. Function
 15482 prototypes shall be provided for use with ISO C standard compilers.

15483 **wint_t** btowc(int);15484 **wint_t** fgetwc(FILE *);15485 **wchar_t** *fgetws(wchar_t *restrict, int, FILE *restrict);15486 **wint_t** fputwc(wchar_t, FILE *);15487 **int** fputws(const wchar_t *restrict, FILE *restrict);

Headers

<wchar.h>

```

15488     int          fwide(FILE *, int);
15489     int          fwprintf(FILE *restrict, const wchar_t *restrict, ...);
15490     int          fwscanf(FILE *restrict, const wchar_t *restrict, ...);
15491     wint_t       getwc(FILE *);
15492     wint_t       getwchar(void);
15493     OB XSI      int          iswalnum(wint_t);
15494     int          iswalpna(wint_t);
15495     int          iswcntrl(wint_t);
15496     int          iswctype(wint_t, wctype_t);
15497     int          iswdigit(wint_t);
15498     int          iswgraph(wint_t);
15499     int          iswlower(wint_t);
15500     int          iswprint(wint_t);
15501     int          iswpunct(wint_t);
15502     int          iswspace(wint_t);
15503     int          iswupper(wint_t);
15504     int          iswxdigit(wint_t);
15505     size_t       mbrlen(const char *restrict, size_t, mbstate_t *restrict);
15506     size_t       mbrtowc(wchar_t *restrict, const char *restrict, size_t,
15507                          mbstate_t *restrict);
15508     int          mbsinit(const mbstate_t *);
15509     CX          size_t       mbsnrtowcs(wchar_t *restrict, const char **restrict,
15510                                         size_t, size_t, mbstate_t *restrict);
15511     size_t       mbsrtowcs(wchar_t *restrict, const char **restrict, size_t,
15512                          mbstate_t *restrict);
15513     CX          FILE        *open_wmemstream(wchar_t **, size_t *);
15514     wint_t       putwc(wchar_t, FILE *);
15515     wint_t       putwchar(wchar_t);
15516     int          swprintf(wchar_t *restrict, size_t,
15517                          const wchar_t *restrict, ...);
15518     int          swscanf(const wchar_t *restrict,
15519                          const wchar_t *restrict, ...);
15520     OB XSI      wint_t       towlower(wint_t);
15521     wint_t       towupper(wint_t);
15522     wint_t       ungetwc(wint_t, FILE *);
15523     int          vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
15524     int          vfwscanf(FILE *restrict, const wchar_t *restrict, va_list);
15525     int          vswprintf(wchar_t *restrict, size_t,
15526                          const wchar_t *restrict, va_list);
15527     int          vswscanf(const wchar_t *restrict, const wchar_t *restrict,
15528                          va_list);
15529     int          vwprintf(const wchar_t *restrict, va_list);
15530     int          vwscanf(const wchar_t *restrict, va_list);
15531     CX          wchar_t     *wcpcpy(wchar_t restrict*, const wchar_t *restrict);
15532     wchar_t     *wcpncpy(wchar_t restrict*, const wchar_t *restrict, size_t);
15533     size_t       wcrtoomb(char *restrict, wchar_t, mbstate_t *restrict);
15534     CX          int          wcscasecmp(const wchar_t *, const wchar_t *);
15535     int          wcscasecmp_l(const wchar_t *, const wchar_t *, locale_t);
15536     wchar_t     *wcscat(wchar_t *restrict, const wchar_t *restrict);
15537     wchar_t     *wcschr(const wchar_t *, wchar_t);
15538     int          wcscmp(const wchar_t *, const wchar_t *);
15539     int          wcscoll(const wchar_t *, const wchar_t *);

```

<wchar.h>

Headers

15540	CX	int	wscoll_l(const wchar_t *, const wchar_t *, locale_t);
15541		wchar_t	*wcscpy(wchar_t *restrict, const wchar_t *restrict);
15542		size_t	wcscspn(const wchar_t *, const wchar_t *);
15543	CX	wchar_t	*wcsdup(const wchar_t *);
15544		size_t	wcsftime(wchar_t *restrict, size_t,
15545			const wchar_t *restrict, const struct tm *restrict);
15546		size_t	wcslen(const wchar_t *);
15547	CX	int	wcsncasecmp(const wchar_t *, const wchar_t *, size_t);
15548		int	wcsncasecmp_l(const wchar_t *, const wchar_t *, size_t,
15549			locale_t);
15550		wchar_t	*wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
15551		int	wcsncmp(const wchar_t *, const wchar_t *, size_t);
15552		wchar_t	*wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15553	CX	size_t	wcsnlen(const wchar_t *, size_t);
15554		size_t	wcsnrtombs(char *restrict, const wchar_t **restrict, size_t,
15555			size_t, mbstate_t *restrict);
15556		wchar_t	*wcsprbr(const wchar_t *, const wchar_t *);
15557		wchar_t	*wcsrchr(const wchar_t *, wchar_t);
15558		size_t	wcsrtombs(char *restrict, const wchar_t **restrict,
15559			size_t, mbstate_t *restrict);
15560		size_t	wcsspn(const wchar_t *, const wchar_t *);
15561		wchar_t	*wcsstr(const wchar_t *restrict, const wchar_t *restrict);
15562		double	wcstod(const wchar_t *restrict, wchar_t **restrict);
15563		float	wcstof(const wchar_t *restrict, wchar_t **restrict);
15564		wchar_t	*wcstok(wchar_t *restrict, const wchar_t *restrict,
15565			wchar_t **restrict);
15566		long	wcstol(const wchar_t *restrict, wchar_t **restrict, int);
15567		long double	wcstold(const wchar_t *restrict, wchar_t **restrict);
15568		long long	wcstoll(const wchar_t *restrict, wchar_t **restrict, int);
15569		unsigned long	wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
15570		unsigned long long	
15571			wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
15572	XSI	int	wcswidth(const wchar_t *, size_t);
15573		size_t	wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
15574	CX	size_t	wcsxfrm_l(wchar_t *restrict, const wchar_t *restrict,
15575			size_t, locale_t);
15576		int	wctob(wint_t);
15577	OB XSI	wctype_t	wctype(const char *);
15578	XSI	int	wcwidth(wchar_t);
15579		wchar_t	*wmemchr(const wchar_t *, wchar_t, size_t);
15580		int	wmemcmp(const wchar_t *, const wchar_t *, size_t);
15581		wchar_t	*wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15582		wchar_t	*wmemmove(wchar_t *, const wchar_t *, size_t);
15583		wchar_t	*wmemset(wchar_t *, wchar_t, size_t);
15584		int	wprintf(const wchar_t *restrict, ...);
15585		int	wscanf(const wchar_t *restrict, ...);

Headers

<wchar.h>

APPLICATION USAGE

The *iswblank()* function was a late addition to the ISO C standard and was introduced at the same time as the ISO C standard introduced <wctype.h>, which contains all of the *isw*()* functions. The Open Group Base Specifications had previously aligned with the MSE working draft and had introduced the rest of the *isw*()* functions into <wchar.h>. For backwards-compatibility, the original set of *isw*()* functions, without *iswblank()*, are permitted (as part of the XSI option) in <wchar.h>. For maximum portability, applications should include <wctype.h> in order to obtain declarations for the *isw*()* functions. This compatibility has been made obsolescent.

RATIONALE

In the ISO C standard, the symbols referenced as XSI extensions are in <wctype.h>. Their presence here is thus an extension.

FUTURE DIRECTIONS

None.

SEE ALSO

<ctype.h>, <locale.h>, <stdarg.h>, <stddef.h>, <stdint.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, <wctype.h>

XSH Section 2.2 (on page 468), *btowc()*, *confstr()*, *fgetwc()*, *fgetws()*, *fputwc()*, *fputws()*, *fwide()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *mbrlen()*, *mbrtowc()*, *mbsinit()*, *mbsrtowcs()*, *open_memstream()*, *putwc()*, *putwchar()*, *towlower()*, *towupper()*, *ungetwc()*, *vwfprintf()*, *vwscanf()*, *wcrtomb()*, *wcscasecmp()*, *wcscat()*, *wcschr()*, *wcscmp()*, *wscoll()*, *wscpy()*, *wcscspn()*, *wcsdup()*, *wcsftime()*, *wcslen()*, *wcsncat()*, *wcsncmp()*, *wcsncpy()*, *wcspbrk()*, *wcsrchr()*, *wcsrtombs()*, *wcsspn()*, *wcsstr()*, *wcstod()*, *wcstok()*, *wcstol()*, *wcstoul()*, *wcswidth()*, *wcsxfrm()*, *wctob()*, *wctype()*, *wcwidth()*, *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemmove()*, *wmemset()*

XCU *getconf*

CHANGE HISTORY

First released in Issue 4.

Issue 5

Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The Open Group Corrigendum U021/10 is applied. The prototypes for *wcswidth()* and *wcwidth()* are marked as extensions.

The Open Group Corrigendum U028/5 is applied, correcting the prototype for the *mbsinit()* function.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- Various function prototypes are updated to add the **restrict** keyword.
- The functions *vwscanf()*, *vsscanf()*, *wcstof()*, *wcstold()*, *wcstoll()*, and *wcstoull()* are added.

The type **wctype_t**, the *isw*()*, *to*()*, and *wctype()* functions are marked as XSI extensions.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/26 is applied, adding the APPLICATION USAGE section.

<wchar.h>

Headers

15629 **Issue 7**

15630 The *mbsnrtowcs()*, *open_wmemstream()*, *wcpcpy()*, *wcpncpy()*, *wscasecmp()*, *wcsdup()*,
15631 *wcsncasecmp()*, *wcsnlen()*, and *wscnrtombs()* functions are added from The Open Group
15632 Technical Standard, 2006, Extended API Set Part 1.

15633 The *wscasecmp_l()*, *wcsncasecmp_l()*, *wscoll_l()*, and *wcsxfrm_l()* functions are added from The
15634 Open Group Technical Standard, 2006, Extended API Set Part 4.

15635 The **wctype_t** type, and the *isw**, *towlower()*, and *towupper()* functions are marked obsolescent in
15636 <wchar.h> since the ISO C standard requires the declarations to be in <wctype.h>.

15637 This reference page is clarified with respect to macros and symbolic constants, and a declaration
15638 for the **locale_t** type is added.

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

Headers

<wctype.h>

NAME

wctype.h — wide-character classification and mapping utilities

SYNOPSIS

```
#include <wctype.h>
```

DESCRIPTION

Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 468) to enable the visibility of these symbols in this header.

The <wctype.h> header shall define the following types:

wint_t As described in <wchar.h>.

wctrans_t A scalar type that can hold values which represent locale-specific character mappings.

wctype_t As described in <wchar.h>.

The <wctype.h> header shall define the **locale_t** type as described in <locale.h>.

The <wctype.h> header shall define the following macro:

WEOF As described in <wchar.h>.

For all functions described in this header that accept an argument of type **wint_t**, the value is representable as a **wchar_t** or equals the value of WEOF. If this argument has any other value, the behavior is undefined.

The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.

Inclusion of the <wctype.h> header may make visible all symbols from the headers <ctype.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, and <wchar.h>.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

```
int      iswalnum(wint_t);
int      iswalnum_l(wint_t, locale_t);
int      iswalpunct(wint_t);
int      iswalpunct_l(wint_t, locale_t);
int      iswblank(wint_t);
int      iswblank_l(wint_t, locale_t);
int      iswcntrl(wint_t);
int      iswcntrl_l(wint_t, locale_t);
int      iswctype(wint_t, wctype_t);
int      iswctype_l(wint_t, wctype_t, locale_t);
int      iswdigit(wint_t);
int      iswdigit_l(wint_t, locale_t);
int      iswgraph(wint_t);
int      iswgraph_l(wint_t, locale_t);
int      iswlower(wint_t);
int      iswlower_l(wint_t, locale_t);
int      iswprint(wint_t);
int      iswprint_l(wint_t, locale_t);
int      iswpunct(wint_t);
int      iswpunct_l(wint_t, locale_t);
int      iswspace(wint_t);
```

<wctype.h>

Headers

```

15684 CX      int      iswspace_l(wint_t, locale_t);
15685          int      iswupper(wint_t);
15686 CX      int      iswupper_l(wint_t, locale_t);
15687          int      iswxdigit(wint_t);
15688 CX      int      iswxdigit_l(wint_t, locale_t);
15689          wint_t    towctrans(wint_t, wctrans_t);
15690 CX      wint_t    towctrans_l(wint_t, wctrans_t, locale_t);
15691          wint_t    tolower(wint_t);
15692 CX      wint_t    tolower_l(wint_t, locale_t);
15693          wint_t    toupper(wint_t);
15694 CX      wint_t    toupper_l(wint_t, locale_t);
15695          wctrans_t wctrans(const char *);
15696 CX      wctrans_t wctrans_l(const char *, locale_t);
15697          wctype_t  wctype(const char *);
15698 CX      wctype_t  wctype_l(const char *, locale_t);

```

15699 APPLICATION USAGE

15700 None.

15701 RATIONALE

15702 None.

15703 FUTURE DIRECTIONS

15704 None.

15705 SEE ALSO

15706 <ctype.h>, <locale.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>,
 15707 <wchar.h>

15708 XSH Section 2.2 (on page 468), *iswalnum()*, *iswalph()*, *iswblank()*, *iswcntrl()*, *iswctype()*,
 15709 *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*,
 15710 *setlocale()*, *towctrans()*, *towlower()*, *toupper()*, *wctrans()*, *wctype()*

15711 CHANGE HISTORY

15712 First released in Issue 5. Derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15713 Issue 6

15714 The *iswblank()* function is added for alignment with the ISO/IEC 9899:1999 standard.

15715 Issue 7

15716 SD5-XBD-ERN-6 is applied.

15717 The **l()* functions are added from The Open Group Technical Standard, 2006, Extended API Set
 15718 Part 4.

15719 This reference page is clarified with respect to macros and symbolic constants.

Headers

<wordexp.h>

NAME

wordexp.h — word-expansion types

SYNOPSIS

```
#include <wordexp.h>
```

DESCRIPTION

The <wordexp.h> header shall define the structures and symbolic constants used by the *wordexp()* and *wordfree()* functions.

The <wordexp.h> header shall define the **wordexp_t** structure type, which shall include at least the following members:

size_t	we_wordc	Count of words matched by <i>words</i> .
char	**we_wordv	Pointer to list of expanded words.
size_t	we_offs	Slots to reserve at the beginning of <i>we_wordv</i> .

The <wordexp.h> header shall define the following symbolic constants for use as flags for the *wordexp()* function:

WRDE_APPEND	Append words to those previously generated.
WRDE_DOOFFS	Number of null pointers to prepend to <i>we_wordv</i> .
WRDE_NOCMD	Fail if command substitution is requested.
WRDE_REUSE	The <i>pwordexp</i> argument was passed to a previous successful call to <i>wordexp()</i> , and has not been passed to <i>wordfree()</i> . The result is the same as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i> without WRDE_REUSE.

WRDE_SHOWERR	Do not redirect <i>stderr</i> to <i>/dev/null</i> .

WRDE_UNDEF	Report error on an attempt to expand an undefined shell variable.

The <wordexp.h> header shall define the following symbolic constants as error return values:

WRDE_BADCHAR	One of the unquoted characters—<newline>, ' ', ' & ', ' ; ', ' < ', ' > ', ' (', ') ', ' { ', ' } '—appears in <i>words</i> in an inappropriate context.
WRDE_BADVAL	Reference to undefined shell variable when WRDE_UNDEF is set in <i>flags</i> .
WRDE_CMDSUB	Command substitution requested when WRDE_NOCMD was set in <i>flags</i> .
WRDE_NOSPACE	Attempt to allocate memory failed.
WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated string.

The <wordexp.h> header shall define the **size_t** type as described in <stddef.h>.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int wordexp(const char *restrict, wordexp_t *restrict, int);
void wordfree(wordexp_t *);
```

<wordexp.h>

Headers

15756 APPLICATION USAGE

15757 None.

15758 RATIONALE

15759 None.

15760 FUTURE DIRECTIONS

15761 None.

15762 SEE ALSO

15763 <stddef.h>

15764 XSH Section 2.6

15765 CHANGE HISTORY

15766 First released in Issue 4. Derived from the ISO POSIX-2 standard.

15767 Issue 6

15768 The **restrict** keyword is added to the prototype for *wordexp()*.

15769 The WRDE_NOSYS constant is marked obsolescent.

15770 Issue 7

15771 The obsolescent WRDE_NOSYS constant is removed.

15772 This reference page is clarified with respect to macros and symbolic constants.

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

15773

Technical Standard

15774

Vol. 2:

15775

System Interfaces, Issue 7

15776

The Open Group

15777

The Institute of Electrical and Electronics Engineers, Inc.

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

(blank page)

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

Introduction

The System Interfaces volume of POSIX.1-2008 describes the interfaces offered to application programs by POSIX-conformant systems.

1.1 Relationship to Other Formal Standards

Great care has been taken to ensure that this volume of POSIX.1-2008 is fully aligned with the following standards:

ISO C (1999)

ISO/IEC 9899:1999, Programming Languages — C, including ISO/IEC 9899:1999/Cor.1:2001(E), ISO/IEC 9899:1999/Cor.2:2004(E), and ISO/IEC 9899:1999/Cor.3.

Parts of the ISO/IEC 9899:1999 standard (hereinafter referred to as the ISO C standard) are referenced to describe requirements also mandated by this volume of POSIX.1-2008. Some functions and headers included within this volume of POSIX.1-2008 have a version in the ISO C standard; in this case CX markings are added as appropriate to show where the ISO C standard has been extended (see [Section 1.7.1](#), on page 7). Any conflict between this volume of POSIX.1-2008 and the ISO C standard is unintentional.

This volume of POSIX.1-2008 also allows, but does not require, mathematics functions to support IEEE Std 754-1985 and IEEE Std 854-1987.

1.2 Format of Entries

The entries in [Chapter 3](#) are based on a common format as follows. The only sections relating to conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections.

NAME

This section gives the name or names of the entry and briefly states its purpose.

SYNOPSIS

This section summarizes the use of the entry being described. If it is necessary to include a header to use this function, the names of such headers are shown, for example:

```
#include <stdio.h>
```

DESCRIPTION

This section describes the functionality of the function or header.

RETURN VALUE

This section indicates the possible return values, if any.

If the implementation can detect errors, “successful completion” means that no error

- 15812 has been detected during execution of the function. If the implementation does detect
15813 an error, the error is indicated.
- 15814 For functions where no errors are defined, “successful completion” means that if the
15815 implementation checks for errors, no error has been detected. If the implementation can
15816 detect errors, and an error is detected, the indicated return value is returned and *errno*
15817 may be set.
- 15818 **ERRORS**
- 15819 This section gives the symbolic names of the error values returned by a function or
15820 stored into a variable accessed through the symbol *errno* if an error occurs.
- 15821 “No errors are defined” means that error values returned by a function or stored into a
15822 variable accessed through the symbol *errno*, if any, depend on the implementation.
- 15823 **EXAMPLES**
- 15824 This section is informative.
- 15825 This section gives examples of usage, where appropriate. In the event of conflict
15826 between an example and a normative part of this volume of POSIX.1-2008, the
15827 normative material is to be taken as correct.
- 15828 **APPLICATION USAGE**
- 15829 This section is informative.
- 15830 This section gives warnings and advice to application developers about the entry. In the
15831 event of conflict between warnings and advice and a normative part of this volume of
15832 POSIX.1-2008, the normative material is to be taken as correct.
- 15833 **RATIONALE**
- 15834 This section is informative.
- 15835 This section contains historical information concerning the contents of this volume of
15836 POSIX.1-2008 and why features were included or discarded by the standard
15837 developers.
- 15838 **FUTURE DIRECTIONS**
- 15839 This section is informative.
- 15840 This section provides comments which should be used as a guide to current thinking;
15841 there is not necessarily a commitment to adopt these future directions.
- 15842 **SEE ALSO**
- 15843 This section is informative.
- 15844 This section gives references to related information.
- 15845 **CHANGE HISTORY**
- 15846 This section is informative.
- 15847 This section shows the derivation of the entry and any significant changes that have
15848 been made to it.

General Information

This chapter covers information that is relevant to all the functions specified in [Chapter 3](#) and XBD [Chapter 13](#) (on page 219).

2.1 Use and Implementation of Interfaces

2.1.1 Use and Implementation of Functions

Each of the following statements shall apply to all functions unless explicitly stated otherwise in the detailed descriptions that follow:

1. If an argument to a function has an invalid value (such as a value outside the domain of the function, or a pointer outside the address space of the program, or a null pointer), the behavior is undefined.
2. Any function declared in a header may also be implemented as a macro defined in the header, so a function should not be declared explicitly if its header is included. Any macro definition of a function can be suppressed locally by enclosing the name of the function in parentheses, because the name is then not followed by the <left-parenthesis> that indicates expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a function even if it is also defined as a macro. The use of the C-language `#undef` construct to remove any such macro definition shall also ensure that an actual function is referred to.
3. Any invocation of a function that is implemented as a macro shall expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so it is generally safe to use arbitrary expressions as arguments.
4. Provided that a function can be declared without reference to any type defined in a header, it is also permissible to declare the function explicitly and use it without including its associated header.
5. If a function that accepts a variable number of arguments is not declared (explicitly or by including its associated header), the behavior is undefined.

2.1.2 Use and Implementation of Macros

Each of the following statements shall apply to all macros unless explicitly stated otherwise:

1. Any definition of an object-like macro in a header shall expand to code that is fully protected by parentheses where necessary, so that it groups in an arbitrary expression as if it were a single identifier.
2. All object-like macros listed as expanding to integer constant expressions shall additionally be suitable for use in `#if` preprocessing directives.
3. Any definition of a function-like macro in a header shall expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so that it is generally safe to use arbitrary expressions as arguments.
4. Any definition of a function-like macro in a header can be invoked in an expression anywhere a function with a compatible return type could be called.

2.2 The Compilation Environment

2.2.1 POSIX.1 Symbols

Certain symbols in this volume of POSIX.1-2008 are defined in headers (see XBD [Chapter 13](#), on page 219). Some of those headers could also define symbols other than those defined by POSIX.1-2008, potentially conflicting with symbols used by the application. Also, POSIX.1-2008 defines symbols that are not permitted by other standards to appear in those headers without some control on the visibility of those symbols.

Symbols called “feature test macros” are used to control the visibility of symbols that might be included in a header. Implementations, future versions of this standard, and other standards may define additional feature test macros.

In the compilation of an application that **#defines** a feature test macro specified by POSIX.1-2008, no header defined by POSIX.1-2008 shall be included prior to the definition of the feature test macro. This restriction also applies to any implementation-provided header in which these feature test macros are used. If the definition of the macro does not precede the **#include**, the result is undefined.

Feature test macros shall begin with the <underscore> character ('_').

2.2.1.1 The `_POSIX_C_SOURCE` Feature Test Macro

A POSIX-conforming application shall ensure that the feature test macro `_POSIX_C_SOURCE` is defined before inclusion of any header.

When an application includes a header described by POSIX.1-2008, and when this feature test macro is defined to have the value 200809L:

1. All symbols required by POSIX.1-2008 to appear when the header is included shall be made visible.

General Information

The Compilation Environment

2. Symbols that are explicitly permitted, but not required, by POSIX.1-2008 to appear in that header (including those in reserved name spaces) may be made visible.

3. Additional symbols not required or explicitly permitted by POSIX.1-2008 to be in that header shall not be made visible, except when enabled by another feature test macro.

Identifiers in POSIX.1-2008 may only be undefined using the **#undef** directive as described in [Section 2.1](#) (on page 467) or [Section 2.2.2](#). These **#undef** directives shall follow all **#include** directives of any header in POSIX.1-2008.

Note: The POSIX.1-1990 standard specified a macro called `_POSIX_SOURCE`. This has been superseded by `_POSIX_C_SOURCE`.

2.2.1.2 The `_XOPEN_SOURCE` Feature Test Macro

An XSI-conforming application shall ensure that the feature test macro `_XOPEN_SOURCE` is defined with the value 700 before inclusion of any header. This is needed to enable the functionality described in [Section 2.2.1.1](#) (on page 468) and to ensure that the XSI option is enabled.

Since this volume of POSIX.1-2008 is aligned with the ISO C standard, and since all functionality enabled by `_POSIX_C_SOURCE` set equal to 200809L is enabled by `_XOPEN_SOURCE` set equal to 700, there should be no need to define `_POSIX_C_SOURCE` if `_XOPEN_SOURCE` is so defined. Therefore, if `_XOPEN_SOURCE` is set equal to 700 and `_POSIX_C_SOURCE` is set equal to 200809L, the behavior is the same as if only `_XOPEN_SOURCE` is defined and set equal to 700. However, should `_POSIX_C_SOURCE` be set to a value greater than 200809L, the behavior is unspecified.

If `_XOPEN_SOURCE` is defined with the value 700 and `_POSIX_C_SOURCE` is undefined before inclusion of any header, then the header may define the `_POSIX_C_SOURCE` macro with the value 200809L.

2.2.2 The Name Space

All identifiers in this volume of POSIX.1-2008, except *environ*, are defined in at least one of the headers, as shown in [XBD Chapter 13](#) (on page 219). When `_XOPEN_SOURCE` or `_POSIX_C_SOURCE` is defined, each header defines or declares some identifiers, potentially conflicting with identifiers used by the application. The set of identifiers visible to the application consists of precisely those identifiers from the header pages of the included headers, as well as additional identifiers reserved for the implementation. In addition, some headers may make visible identifiers from other headers as indicated on the relevant header pages.

Implementations may also add members to a structure or union without controlling the visibility of those members with a feature test macro, as long as a user-defined macro with the same name cannot interfere with the correct interpretation of the program. The identifiers reserved for use by the implementation are described below:

1. Each identifier with external linkage described in the header section is reserved for use as an identifier with external linkage if the header is included.
2. Each macro described in the header section is reserved for any use if the header is included.
3. Each identifier with file scope described in the header section is reserved for use as an identifier with file scope in the same name space if the header is included.

The prefixes `posix_`, `POSIX_`, and `_POSIX_` are reserved for use by POSIX.1-2008 and other

- 15954 POSIX standards. Implementations may add symbols to the headers shown in the following
15955 table, provided the identifiers for those symbols either:
- 15956 1. Begin with the corresponding reserved prefixes in the table, or
 - 15957 2. Have one of the corresponding complete names in the table, or
 - 15958 3. End in the string indicated as a reserved suffix in the table and do not use the reserved
15959 prefixes `posix_`, `POSIX_`, or `_POSIX_`, as long as the reserved suffix is in that part of the
15960 name considered significant by the implementation.
- 15961 Symbols that use the reserved prefix `_POSIX_` may be made visible by implementations in any
15962 header defined by POSIX.1-2008.

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

General Information

The Compilation Environment

		Header	Prefix	Suffix	Complete Name
15963		<aio.h>	aio_, lio_, AIO_, LIO_		
15964		<arpa/inet.h>	inet_		
15965		<ctype.h>	to[a-z], is[a-z]		
15966		<dlfcn.h>	RTLD_		
15967		<dirent.h>	d_		
15968		<fcntl.h>	l_		
15969		<fmtmsg.h>	MM_		
15970	XSI	<fnmatch.h>	FNM_		
15971	XSI	<ftw.h>	FTW		
15972		<glob.h>	gl_, GLOB_		
15973		<grp.h>	gr_		
15974		<limits.h>		_MAX, _MIN	
15975	MSG	<mqueue.h>	mq_, MQ_		
15976	XSI	<ndbm.h>	dbm_, DBM_		
15977		<netdb.h>	ai_, h_, n_, p_, s_		
15978		<net/if.h>	if_, IF_		
15979		<netinet/in.h>	in_, ip_, s_, sin_, INADDR_, IPPROTO_		
15980	IP6		in6_, s6_, sin6_, IPV6_		
15981		<netinet/tcp.h>	TCP_		
15982		<nl_types.h>	NL_		
15983		<poll.h>	pd_, ph_, ps_, POLL		
15984		<pthread.h>	pthread_, PTHREAD_		
15985		<pwd.h>	pw_		
15986		<regex.h>	re_, rm_, REG_		
15987		<sched.h>	sched_, SCHED_		
15988		<semaphore.h>	sem_, SEM_		
15989		<signal.h>	sa_, si_, sigev_, sival_, uc_, BUS_, CLD_, FPE_, ILL_, SA_, SEGV_, SI_, SIGEV_,		
15990	XSI		ss_, sv_, SS_, TRAP_		
15991	OB XSR	<stropts.h>	POLL_		
15992			bi_, ic_, l_, sl_, str_, FLUSH[A-Z], I_, S_, SND[A-Z]		
15993		<stdint.h>			int[0-9a-z]*_t, uint[0-9a-z]*_t
15994		<stdlib.h>	str[a-z]		
15995		<string.h>	str[a-z], mem[a-z], wcs[a-z]		
16000	XSI	<sys/ipc.h>	ipc_, IPC_		key, pad, seq
16001		<sys/mman.h>	shm_, MAP_, MCL_, MS_, PROT_		
16002	XSI	<sys/msg.h>	msg, MSG_[A-Z]		msg
16003	XSI	<sys/resource.h>	rlim_, ru_, PRIO_, RLIMIT_, RUSAGE_		
16004		<sys/select.h>	fd_, fds_, FD_		

		Header	Prefix	Suffix	Complete Name
16007					
16008					
16009	XSI	<sys/sem.h>	sem, SEM_		sem
16010	XSI	<sys/shm.h>	shm, SHM[A-Z], SHM_[A-Z]		
16011		<sys/socket.h>	cmmsg_, if_, ifc_, ifra_, ifru_, infu_, l_, msg_, sa_, ss_, AF_, MSG_, PF_, SCM_, SHUT_, SO		
16012					
16013	XSI				
16014					
16015		<sys/stat.h>	st_		
16016		<sys/statvfs.h>	f_, ST_		
16017	XSI	<sys/time.h>	it_, tv_, ITIMER_		
16018		<sys/times.h>	tms_		
16019	XSI	<sys/uio.h>	iov_		UIO_MAXIOV
16020		<sys/un.h>	sun_		
16021		<sys/utsname.h>	uts_		
16022		<sys/wait.h>	P_, W[A-Z]		
16023	XSI	<syslog.h>	LOG_		
16024		<termios.h>	c_, B[0-9], TC		
16025		<time.h>	tm_ clock_, it_, timer_, tv_, CLOCK_, TIMER_		
16026					
16027					
16028	XSI	<ulimit.h>	UL_		
16029		<utime.h>	utim_		
16030	XSI	<utmpx.h>	ut_	_LVL, _PROCESS, _TIME	
16031					
16032		<wchar.h>	wcs[a-z]		
16033		<wctype.h>	is[a-z], to[a-z]		
16034		<wordexp.h>	we_, WRDE_		
16035		ANY header		_t	

Note: The notation [A-Z] indicates any uppercase letter in the portable character set. The notation [a-z] indicates any lowercase letter in the portable character set. Commas and spaces in the lists of prefixes and complete names in the above table are not part of any prefix or complete name.

General Information

The Compilation Environment

If any header in the following table is included, macros with the prefixes shown may be defined. After the last inclusion of a given header, an application may use identifiers with the corresponding prefixes for its own purpose, provided their use is preceded by a **#undef** of the corresponding macro.

Header	Prefix
<errno.h>	E[0-9], E[A-Z]
<fcntl.h>	F_, O_
<fenv.h>	FE_[A-Z]
<inttypes.h>	PRI[Xa-z], SCN[Xa-z]
<locale.h>	LC_[A-Z]
<math.h>	FP_[A-Z]
<netinet/in.h>	IMPLINK_, IN_, IP_, IPPORT_, SOCK_,
	IN6_
<signal.h>	SIG_, SIG[A-Z],
	SV_
<stdio.h>	SEEK_
<stropts.h>	M_, MUXID_R[A-Z], STR
<sys/resource.h>	RLIM_
<sys/socket.h>	CMSG_
<sys/stat.h>	S_
<sys/uio.h>	IOV_
<termios.h>	I, O, V (See below.)
<unistd.h>	SEEK_

The following are used to reserve complete names for the **<stdint.h>** header:

INT[0-9A-Za-z-]*_MIN
 INT[0-9A-Za-z-]*_MAX
 INT[0-9A-Za-z-]*_C
 UINT[0-9A-Za-z-]*_MIN
 UINT[0-9A-Za-z-]*_MAX
 UINT[0-9A-Za-z-]*_C

Note: The notation [0-9] indicates any digit. The notation [A-Z] indicates any uppercase letter in the portable character set. The notation [0-9a-z-] indicates any digit, any lowercase letter in the portable character set, or <underscore>.

The following reserved names are used as exact matches for **<termios.h>**:

CBAUD	EXTB	VDSUSP
DEFECHO	FLUSHO	VLNEXT
ECHOCTL	LOBLK	VREPRINT
ECHOKE	PENDIN	VSTATUS
ECHOPRT	SWTCH	VWERASE
EXTA	VDISCARD	

- 16079 The following identifiers are reserved regardless of the inclusion of headers:
- 16080 1. With the exception of identifiers beginning with the prefix `_POSIX_`, all identifiers that
- 16081 begin with an `<underscore>` and either an uppercase letter or another `<underscore>` are
- 16082 always reserved for any use by the implementation.
- 16083 2. All identifiers that begin with an `<underscore>` are always reserved for use as identifiers
- 16084 with file scope in both the ordinary identifier and tag name spaces.
- 16085 3. All identifiers in the table below are reserved for use as identifiers with external linkage.
- 16086 Some of these identifiers do not appear in this volume of POSIX.1-2008, but are reserved for
- 16087 future use by the ISO C standard.
- 16088 4. All functions and external identifiers defined in XBD [Chapter 13](#) (on page 219) are reserved
- 16089 for use as identifiers with external linkage.
- 16090 5. All the identifiers defined in this volume of POSIX.1-2008 that have external linkage are
- 16091 always reserved for use as identifiers with external linkage.
- 16092 **Note:** The notation `[a-z]` indicates any lowercase letter in the portable character set. The notation `'*'`
- 16093 indicates any combination of digits, letters in the portable character set, or `<underscore>`.
- 16094 No other identifiers are reserved.

General Information

The Compilation Environment

16095	_Exit	casinh	clog10	erfcl	fminl
16096	abort	casinhf	clog10f	erff	fmod
16097	abs	casinhl	clog10l	erfl	fmodf
16098	acos	casinl	clog1p	errno	fmodl
16099	acosf	catan	clog1pf	exit	fopen
16100	acosh	catanf	clog1pl	exp	fprintf
16101	acoshf	catanh	clog2	exp2	fputc
16102	acoshl	catanh	clog2f	exp2f	fputs
16103	acosl	catanhf	clog2l	exp2l	fputwc
16104	acosl	catanhf	clogf	expf	fputws
16105	asctime	catanhl	clogl	expl	fread
16106	asin	catanhl	conj	expm1	free
16107	asinf	catanl	conjf	expm1f	freopen
16108	asinh	cbrt	conjl	expm1l	frexp
16109	asinhf	cbrtf	copysign	fabs	frexpf
16110	asinhf	cbrtl	copysignf	fabsf	frexpl
16111	asinl	ccos	copysignl	fabsl	fscanf
16112	asinl	ccosf	cos	fclose	fseek
16113	atan	ccosh	cosf	fdim	fsetpos
16114	atan2	ccoshf	cosh	fdimf	ftell
16115	atan2f	ccoshl	coshf	fdiml	fwide
16116	atan2l	ccosl	coshl	feclearexcept	fwprintf
16117	atanf	ceil	cosl	fegetenv	fwrite
16118	atanf	ceilf	cpow	fegetexceptflag	fwscanf
16119	atanh	ceilf	cpowf	fegetround	getc
16120	atanh	ceill	cpowl	feholdexcept	getchar
16121	atanhf	ceill	cproj	feof	getenv
16122	atanhl	cerf	cprojf	feraiseexcept	gets
16123	atanl	cerfc	cprojl	ferror	getwc
16124	atanl	cerfcf	creal	fesetenv	getwchar
16125	atexit	cerfcl	crealf	fesetexceptflag	gmtime
16126	atof	cerff	creall	fesetround	hypotf
16127	atoi	cerfl	csin	fetestexcept	hypotl
16128	atol	cexpm1	csinf	feupdateenv	ilogb
16129	atoll	cexpm1f	csinh	fflush	ilogbf
16130	bsearch	cexpm1l	csinhf	fgetc	ilogbl
16131	cabs	cexp	csinhf	fgetpos	imaxabs
16132	cabsf	cexp2	csinl	fgets	imaxdiv
16133	cabsf	cexp2f	csqrt	fgetwc	is[a-z]*
16134	cacos	cexp2l	csqrtf	fgetws	isblank
16135	cacosf	cexpf	csqrtl	floor	iswblank
16136	cacosh	cexpl	ctan	floorf	labs
16137	cacoshf	cimag	ctanf	floorl	ldexp
16138	cacoshl	cimagf	ctanl	fma	ldexpf
16139	cacosl	cimagl	ctgamma	fmaf	ldexpl
16140	calloc	clearerr	ctgammaf	fmax	ldiv
16141	carg	clgamma	ctgamma	fmaxf	ldiv
16142	cargf	clgammaf	ltime	fmaxl	lgammaf
16143	cargl	clgamma	difftime	fmaxl	lgamma
16144	casin	clock	div	fmin	llabs
16145	casinf	clog	erfcf	fminf	llrint

16146	llrintf	mbsinit	realloc	sprintf	ungetwc
16147	llrintl	mbsrtowcs	remainderf	sqrt	va_end
16148	llround	mbstowcs	remainderl	sqrtrf	vfprintf
16149	llroundf	mbtowc	remove	sqrtrl	vfscanf
16150	llroundl	mem[a-z]*	remquo	srand	vfwprintf
16151	localeconv	mktime	remquof	sscanf	vfwscanf
16152	localtime	modf	remquol	str[a-z]*	vprintf
16153	log	modff	rename	strtof	vscanf
16154	log10	modfl	rewind	strtoimax	vsprintf
16155	log10f	nan	rint	strtold	vsscanf
16156	log10l	nanf	rintf	strtoll	vswprintf
16157	log1p	nanl	rintl	strtoull	vswscanf
16158	log1pf	nearbyint	round	strtoumax	vwprintf
16159	log1pl	nearbyintf	roundf	swprintf	vwscanf
16160	log2	nearbyintl	roundl	swscanf	wcrtomb
16161	log2f	nextafterf	scalbln	system	wcs[a-z]*
16162	log2l	nextafterl	scalblnf	tan	wcstof
16163	logb	nexttoward	scalblnl	tanf	wcstoimax
16164	logbf	nexttowardf	scalbn	tanh	wcstold
16165	logbl	nexttowardl	scalbnf	tanhf	wcstoll
16166	logf	perror	scalbnl	tanhf	wcstoull
16167	logl	pow	scanf	tanl	wcstoumax
16168	longjmp	powf	setbuf	tgamma	wctob
16169	lrint	powl	setjmp	tgammaf	wctomb
16170	lrintf	printf	setlocale	tgammaf	wctrans
16171	lrintl	putc	setvbuf	time	wctype
16172	lround	putchar	signal	tmpfile	wcwidth
16173	lroundf	puts	sin	tmpnam	wmem[a-z]*
16174	lroundl	putwc	sinf	to[a-z]*	wprintf
16175	malloc	putwchar	sinh	trunc	wscanf
16176	mblen	qsort	sinhf	truncf	
16177	mbrlen	raise	sinhl	truncl	
16178	mbrtowc	rand	sinl	ungetc	

Applications shall not declare or define identifiers with the same name as an identifier reserved in the same context. Since macro names are replaced whenever found, independent of scope and name space, macro names matching any of the reserved identifier names shall not be defined by an application if any associated header is included.

Except that the effect of each inclusion of `<assert.h>` depends on the definition of `NDEBUG`, headers may be included in any order, and each may be included more than once in a given scope, with no difference in effect from that of being included only once.

If used, the application shall ensure that a header is included outside of any external declaration or definition, and it shall be first included before the first reference to any type or macro it defines, or to any function or object it declares. However, if an identifier is declared or defined in more than one header, the second and subsequent associated headers may be included after the initial reference to the identifier. Prior to the inclusion of a header, the application shall not define any macros with names lexically identical to symbols defined by that header.

2.3 Error Numbers

Most functions can provide an error number. The means by which each function provides its error numbers is specified in its description.

Some functions provide the error number in a variable accessed through the symbol *errno*, defined by including the `<errno.h>` header. The value of *errno* should only be examined when it is indicated to be valid by a function's return value. No function in this volume of POSIX.1-2008 shall set *errno* to zero. For each thread of a process, the value of *errno* shall not be affected by function calls or assignments to *errno* by other threads.

Some functions return an error number directly as the function value. These functions return a value of zero to indicate success.

If more than one error occurs in processing a function call, any one of the possible errors may be returned, as the order of detection is undefined.

Implementations may support additional errors not included in this list, may generate errors included in this list under circumstances other than those described here, or may contain extensions or limitations that prevent some errors from occurring.

The ERRORS section on each reference page specifies which error conditions shall be detected by all implementations ("shall fail") and which may be optionally detected by an implementation ("may fail"). If no error condition is detected, the action requested shall be successful.

Implementations may generate error numbers listed here under circumstances other than those described, if and only if all those error conditions can always be treated identically to the error conditions as described in this volume of POSIX.1-2008. Implementations shall not generate a different error number from one required by this volume of POSIX.1-2008 for an error condition described in this volume of POSIX.1-2008, but may generate additional errors unless explicitly disallowed for a particular function.

Each implementation shall document, in the conformance document, situations in which each of the optional conditions defined in POSIX.1-2008 is detected. The conformance document may also contain statements that one or more of the optional error conditions are not detected.

Certain threads-related functions are not allowed to return an error code of [EINTR]. Where this applies it is stated in the ERRORS section on the individual function pages.

The following macro names identify the possible error numbers, in the context of the functions specifically defined in this volume of POSIX.1-2008; these general descriptions are more precisely defined in the ERRORS sections of the functions that return them. Only these macro names should be used in programs, since the actual value of the error number is unspecified. All values listed in this section shall be unique, except as noted below. The values for all these macros shall be found in the `<errno.h>` header defined in the Base Definitions volume of POSIX.1-2008. The actual values are unspecified by this volume of POSIX.1-2008.

[E2BIG]

Argument list too long. The sum of the number of bytes used by the new process image's argument list and environment list is greater than the system-imposed limit of {ARG_MAX} bytes.

or:

Lack of space in an output buffer.

or:

Argument is greater than the system-imposed maximum.

16237	[EACCES]	
16238		Permission denied. An attempt was made to access a file in a way forbidden by its file
16239		access permissions.
16240	[EADDRINUSE]	
16241		Address in use. The specified address is in use.
16242	[EADDRNOTAVAIL]	
16243		Address not available. The specified address is not available from the local system.
16244	[EAFNOSUPPORT]	
16245		Address family not supported. The implementation does not support the specified address
16246		family, or the specified address is not a valid address for the address family of the specified
16247		socket.
16248	[EAGAIN]	
16249		Resource temporarily unavailable. This is a temporary condition and later calls to the same
16250		routine may complete normally.
16251	[EALREADY]	
16252		Connection already in progress. A connection request is already in progress for the specified
16253		socket.
16254	[EBADF]	
16255		Bad file descriptor. A file descriptor argument is out of range, refers to no open file, or a
16256		read (write) request is made to a file that is only open for writing (reading).
16257	[EBADMSG]	
16258	OB XSR	Bad message. During a <i>read()</i> , <i>getmsg()</i> , <i>getpmsg()</i> , or <i>ioctl()</i> <i>I_RECVFD</i> request to a
16259		STREAMS device, a message arrived at the head of the STREAM that is inappropriate for
16260		the function receiving the message.
16261		<i>read()</i> Message waiting to be read on a STREAM is not a data message.
16262		<i>getmsg()</i> or <i>getpmsg()</i>
16263		A file descriptor was received instead of a control message.
16264		<i>ioctl()</i> Control or data information was received instead of a file descriptor when
16265		<i>I_RECVFD</i> was specified.
16266		or:
16267		Bad Message. The implementation has detected a corrupted message.
16268	[EBUSY]	
16269		Resource busy. An attempt was made to make use of a system resource that is not currently
16270		available, as it is being used by another process in a manner that would have conflicted
16271		with the request being made by this process.
16272	[ECANCELED]	
16273		Operation canceled. The associated asynchronous operation was canceled before
16274		completion.
16275	[ECHILD]	
16276		No child process. A <i>wait()</i> , <i>waitid()</i> , or <i>waitpid()</i> function was executed by a process that
16277		had no existing or unwaited-for child process.
16278	[ECONNABORTED]	
16279		Connection aborted. The connection has been aborted.

General Information

Error Numbers

16280	[ECONNREFUSED]
16281	Connection refused. An attempt to connect to a socket was refused because there was no
16282	process listening or because the queue of connection requests was full and the underlying
16283	protocol does not support retransmissions.
16284	[ECONNRESET]
16285	Connection reset. The connection was forcibly closed by the peer.
16286	[EDEADLK]
16287	Resource deadlock would occur. An attempt was made to lock a system resource that would
16288	have resulted in a deadlock situation.
16289	[EDESTADDRREQ]
16290	Destination address required. No bind address was established.
16291	[EDOM]
16292	Domain error. An input argument is outside the defined domain of the mathematical
16293	function (defined in the ISO C standard).
16294	[EDQUOT]
16295	Reserved.
16296	[EEXIST]
16297	File exists. An existing file was mentioned in an inappropriate context; for example, as a
16298	new link name in the <i>link()</i> function.
16299	[EFAULT]
16300	Bad address. The system detected an invalid address in attempting to use an argument of a
16301	call. The reliable detection of this error cannot be guaranteed, and when not detected may
16302	result in the generation of a signal, indicating an address violation, which is sent to the
16303	process.
16304	[EFBIG]
16305	File too large. The size of a file would exceed the maximum file size of an implementation
16306	or offset maximum established in the corresponding file description.
16307	[EHOSTUNREACH]
16308	Host is unreachable. The destination host cannot be reached (probably because the host is
16309	down or a remote router cannot reach it).
16310	[EIDRM]
16311	Identifier removed. Returned during XSI interprocess communication if an identifier has
16312	been removed from the system.
16313	[EILSEQ]
16314	Illegal byte sequence. A wide-character code has been detected that does not correspond to
16315	a valid character, or a byte sequence does not form a valid wide-character code (defined in
16316	the ISO C standard).
16317	[EINPROGRESS]
16318	Operation in progress. This code is used to indicate that an asynchronous operation has not
16319	yet completed.
16320	or:
16321	O_NONBLOCK is set for the socket file descriptor and the connection cannot be
16322	immediately established.

16323	[EINTR]	
16324		Interrupted function call. An asynchronous signal was caught by the process during the
16325		execution of an interruptible function. If the signal handler performs a normal return, the
16326		interrupted function call may return this condition (see the Base Definitions volume of
16327		POSIX.1-2008, <signal.h>).
16328	[EINVAL]	
16329		Invalid argument. Some invalid argument was supplied; for example, specifying an
16330		undefined signal in a <i>signal()</i> function or a <i>kill()</i> function.
16331	[EIO]	
16332		Input/output error. Some physical input or output error has occurred. This error may be
16333		reported on a subsequent operation on the same file descriptor. Any other error-causing
16334		operation on the same file descriptor may cause the [EIO] error indication to be lost.
16335	[EISCONN]	
16336		Socket is connected. The specified socket is already connected.
16337	[EISDIR]	
16338		Is a directory. An attempt was made to open a directory with write mode specified.
16339	[ELOOP]	
16340		Symbolic link loop. A loop exists in symbolic links encountered during pathname
16341		resolution. This error may also be returned if more than {SYMLOOP_MAX} symbolic links
16342		are encountered during pathname resolution.
16343	[EMFILE]	
16344		File descriptor value too large or too many open streams. An attempt was made to open a
16345	XSI	file descriptor with a value greater than or equal to {OPEN_MAX}, or greater than or equal
16346		to the soft limit RLIMIT_NOFILE for the process (if smaller than {OPEN_MAX}); or an
16347		attempt was made to open more than the maximum number of streams allowed in the
16348		process.
16349	[EMLINK]	
16350		Too many links. An attempt was made to have the link count of a single file exceed
16351		{LINK_MAX}.
16352	[EMSGSIZE]	
16353		Message too large. A message sent on a transport provider was larger than an internal
16354		message buffer or some other network limit.
16355		or:
16356		Inappropriate message buffer length.
16357	[EMULTIHOP]	
16358		Reserved.
16359	[ENAMETOOLONG]	
16360		Filename too long. The length of a pathname exceeds {PATH_MAX} and the
16361		implementation considers this to be an error, or a pathname component is longer than
16362		{NAME_MAX}. This error may also occur when pathname substitution, as a result of
16363		encountering a symbolic link during pathname resolution, results in a pathname string the
16364		size of which exceeds {PATH_MAX}.
16365	[ENETDOWN]	
16366		Network is down. The local network interface used to reach the destination is down.

General Information

Error Numbers

16367		[ENETRESET]
16368		The connection was aborted by the network.
16369		[ENETUNREACH]
16370		Network unreachable. No route to the network is present.
16371		[ENFILE]
16372		Too many files open in system. Too many files are currently open in the system. The system
16373		has reached its predefined limit for simultaneously open files and temporarily cannot
16374		accept requests to open another one.
16375		[ENOBUFS]
16376		No buffer space available. Insufficient buffer resources were available in the system to
16377		perform the socket operation.
16378	OB XSR	[ENODATA]
16379		No message available. No message is available on the STREAM head read queue.
16380		[ENODEV]
16381		No such device. An attempt was made to apply an inappropriate function to a device; for
16382		example, trying to read a write-only device such as a printer.
16383		[ENOENT]
16384		No such file or directory. A component of a specified pathname does not exist, or the
16385		pathname is an empty string.
16386		[ENOEXEC]
16387		Executable file format error. A request is made to execute a file that, although it has
16388		appropriate privileges, is not in the format required by the implementation for executable
16389		files.
16390		[ENOLCK]
16391		No locks available. A system-imposed limit on the number of simultaneous file and record
16392		locks has been reached and no more are currently available.
16393		[ENOLINK]
16394		Reserved.
16395		[ENOMEM]
16396		Not enough space. The new process image requires more memory than is allowed by the
16397		hardware or system-imposed memory management constraints.
16398		[ENOMSG]
16399		No message of the desired type. The message queue does not contain a message of the
16400		required type during XSI interprocess communication.
16401		[ENOPROTOPT]
16402		Protocol not available. The protocol option specified to <i>setsockopt()</i> is not supported by the
16403		implementation.
16404		[ENOSPC]
16405		No space left on a device. During the <i>write()</i> function on a regular file or when extending a
16406		directory, there is no free space left on the device.
16407	OB XSR	[ENOSR]
16408		No STREAM resources. Insufficient STREAMS memory resources are available to perform a
16409		STREAMS-related function. This is a temporary condition; it may be recovered from if other
16410		processes release resources.

16411	OB XSR	[ENOSTR]
16412		Not a STREAM. A STREAM function was attempted on a file descriptor that was not
16413		associated with a STREAMS device.
16414		[ENOSYS]
16415		Function not implemented. An attempt was made to use a function that is not available in
16416		this implementation.
16417		[ENOTCONN]
16418		Socket not connected. The socket is not connected.
16419		[ENOTDIR]
16420		Not a directory. A component of the specified pathname exists, but it is not a directory,
16421		when a directory was expected.
16422		[ENOTEMPTY]
16423		Directory not empty. A directory other than an empty directory was supplied when an
16424		empty directory was expected.
16425		[ENOTRECOVERABLE]
16426		State not recoverable. The state protected by a robust mutex is not recoverable.
16427		[ENOTSOCK]
16428		Not a socket. The file descriptor does not refer to a socket.
16429		[ENOTSUP]
16430		Not supported. The implementation does not support this feature of the Realtime Option
16431		Group.
16432		[ENOTTY]
16433		Inappropriate I/O control operation. A control function has been attempted for a file or
16434		special file for which the operation is inappropriate.
16435		[ENXIO]
16436		No such device or address. Input or output on a special file refers to a device that does not
16437		exist, or makes a request beyond the capabilities of the device. It may also occur when, for
16438		example, a tape drive is not on-line.
16439		[EOPNOTSUPP]
16440		Operation not supported on socket. The type of socket (address family or protocol) does not
16441		support the requested operation. A conforming implementation may assign the same values
16442		for [EOPNOTSUPP] and [ENOTSUP].
16443		[EOVERFLOW]
16444		Value too large to be stored in data type. An operation was attempted which would
16445		generate a value that is outside the range of values that can be represented in the relevant
16446		data type or that are allowed for a given data item.
16447		[EOWNERDEAD]
16448		Previous owner died. The owner of a robust mutex terminated while holding the mutex
16449		lock.
16450		[EPERM]
16451		Operation not permitted. An attempt was made to perform an operation limited to
16452		processes with appropriate privileges or to the owner of a file or other resource.
16453		[EPIPE]
16454		Broken pipe. A write was attempted on a socket, pipe, or FIFO for which there is no process
16455		to read the data.

General Information

Error Numbers

16456	[EPROTO]	
16457	Protocol error. Some protocol error occurred. This error is device-specific, but is generally	
16458	not related to a hardware failure.	
16459	[EPROTONOSUPPORT]	
16460	Protocol not supported. The protocol is not supported by the address family, or the protocol	
16461	is not supported by the implementation.	
16462	[EPROTOTYPE]	
16463	Protocol wrong type for socket. The socket type is not supported by the protocol.	
16464	[ERANGE]	
16465	Result too large or too small. The result of the function is too large (overflow) or too small	
16466	(underflow) to be represented in the available space (defined in the ISO C standard).	
16467	[EROFS]	
16468	Read-only file system. An attempt was made to modify a file or directory on a file system	
16469	that is read-only.	
16470	[ESPIPE]	
16471	Invalid seek. An attempt was made to access the file offset associated with a pipe or FIFO.	
16472	[ESRCH]	
16473	No such process. No process can be found corresponding to that specified by the given	
16474	process ID.	
16475	[ESTALE]	
16476	Reserved.	
16477	OB XSR [ETIME]	
16478	STREAM <i>ioctl()</i> timeout. The timer set for a STREAMS <i>ioctl()</i> call has expired. The cause of	
16479	this error is device-specific and could indicate either a hardware or software failure, or a	
16480	timeout value that is too short for the specific operation. The status of the <i>ioctl()</i> operation is	
16481	unspecified.	
16482	[ETIMEDOUT]	
16483	Connection timed out. The connection to a remote machine has timed out. If the connection	
16484	timed out during execution of the function that reported this error (as opposed to timing	
16485	out prior to the function being called), it is unspecified whether the function has completed	
16486	some or all of the documented behavior associated with a successful completion of the	
16487	function.	
16488	or:	
16489	Operation timed out. The time limit associated with the operation was exceeded before the	
16490	operation completed.	
16491	[ETXTBSY]	
16492	Text file busy. An attempt was made to execute a pure-procedure program that is currently	
16493	open for writing, or an attempt has been made to open for writing a pure-procedure	
16494	program that is being executed.	
16495	[EWOULDBLOCK]	
16496	Operation would block. An operation on a socket marked as non-blocking has encountered	
16497	a situation such as no data available that otherwise would have caused the function to	
16498	suspend execution.	
16499	A conforming implementation may assign the same values for [EWOULDBLOCK] and	
16500	[EAGAIN].	

16501 [EXDEV]
 16502 Improper link. A link to a file on another file system was attempted.

16503 2.3.1 Additional Error Numbers

16504 Additional implementation-defined error numbers may be defined in `<errno.h>`.

16505 2.4 Signal Concepts

16506 2.4.1 Signal Generation and Delivery

16507 A signal is said to be “generated” for (or sent to) a process or thread when the event that causes
 16508 the signal first occurs. Examples of such events include detection of hardware faults, timer
 16509 expiration, signals generated via the **sigevent** structure and terminal activity, as well as
 16510 invocations of the `kill()` and `sigqueue()` functions. In some circumstances, the same event
 16511 generates signals for multiple processes.

16512 At the time of generation, a determination shall be made whether the signal has been generated
 16513 for the process or for a specific thread within the process. Signals which are generated by some
 16514 action attributable to a particular thread, such as a hardware fault, shall be generated for the
 16515 thread that caused the signal to be generated. Signals that are generated in association with a
 16516 process ID or process group ID or an asynchronous event, such as terminal activity, shall be
 16517 generated for the process.

16518 Each process has an action to be taken in response to each signal defined by the system (see
 16519 [Section 2.4.3](#), on page 486). A signal is said to be “delivered” to a process when the appropriate
 16520 action for the process and signal is taken. A signal is said to be “accepted” by a process when the
 16521 signal is selected and returned by one of the `sigwait()` functions.

16522 During the time between the generation of a signal and its delivery or acceptance, the signal is
 16523 said to be “pending”. Ordinarily, this interval cannot be detected by an application. However, a
 16524 signal can be “blocked” from delivery to a thread. If the action associated with a blocked signal
 16525 is anything other than to ignore the signal, and if that signal is generated for the thread, the
 16526 signal shall remain pending until it is unblocked, it is accepted when it is selected and returned
 16527 by a call to the `sigwait()` function, or the action associated with it is set to ignore the signal.
 16528 Signals generated for the process shall be delivered to exactly one of those threads within the
 16529 process which is in a call to a `sigwait()` function selecting that signal or has not blocked delivery
 16530 of the signal. If there are no threads in a call to a `sigwait()` function selecting that signal, and if all
 16531 threads within the process block delivery of the signal, the signal shall remain pending on the
 16532 process until a thread calls a `sigwait()` function selecting that signal, a thread unblocks delivery
 16533 of the signal, or the action associated with the signal is set to ignore the signal. If the action
 16534 associated with a blocked signal is to ignore the signal and if that signal is generated for the
 16535 process, it is unspecified whether the signal is discarded immediately upon generation or
 16536 remains pending.

16537 Each thread has a “signal mask” that defines the set of signals currently blocked from delivery
 16538 to it. The signal mask for a thread shall be initialized from that of its parent or creating thread,
 16539 or from the corresponding thread in the parent process if the thread was created as the result of a
 16540 call to `fork()`. The `pthread_sigmask()`, `sigaction()`, `sigprocmask()`, and `sigsuspend()` functions control
 16541 the manipulation of the signal mask.

The determination of which action is taken in response to a signal is made at the time the signal is delivered, allowing for any changes since the time of generation. This determination is independent of the means by which the signal was originally generated. If a subsequent occurrence of a pending signal is generated, it is implementation-defined as to whether the signal is delivered or accepted more than once in circumstances other than those in which queuing is required. The order in which multiple, simultaneously pending signals outside the range SIGRTMIN to SIGRTMAX are delivered to or accepted by a process is unspecified.

When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process, any pending SIGCONT signals for that process shall be discarded. Conversely, when SIGCONT is generated for a process, all pending stop signals for that process shall be discarded. When SIGCONT is generated for a process that is stopped, the process shall be continued, even if the SIGCONT signal is blocked or ignored. If SIGCONT is blocked and not ignored, it shall remain pending until it is either unblocked or a stop signal is generated for the process.

An implementation shall document any condition not specified by this volume of POSIX.1-2008 under which the implementation generates signals.

2.4.2 Realtime Signal Generation and Delivery

This section describes functionality to support realtime signal generation and delivery.

Some signal-generating functions, such as high-resolution timer expiration, asynchronous I/O completion, interprocess message arrival, and the *sigqueue()* function, support the specification of an application-defined value, either explicitly as a parameter to the function or in a **sigevent** structure parameter. The **sigevent** structure is defined in **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	<i>sigev_notify</i>	Notification type.
int	<i>sigev_signo</i>	Signal number.
union sigval	<i>sigev_value</i>	Signal value.
void(*) (union sigval)	<i>sigev_notify_function</i>	Notification function.
(pthread_attr_t*)	<i>sigev_notify_attributes</i>	Notification attributes.

The *sigev_notify* member specifies the notification mechanism to use when an asynchronous event occurs. This volume of POSIX.1-2008 defines the following values for the *sigev_notify* member:

SIGEV_NONE	No asynchronous notification shall be delivered when the event of interest occurs.
SIGEV_SIGNAL	The signal specified in <i>sigev_signo</i> shall be generated for the process when the event of interest occurs. If the implementation supports the Realtime Signals Extension option and if the SA_SIGINFO flag is set for that signal number, then the signal shall be queued to the process and the value specified in <i>sigev_value</i> shall be the <i>si_value</i> component of the generated signal. If SA_SIGINFO is not set for that signal number, it is unspecified whether the signal is queued and what value, if any, is sent.
SIGEV_THREAD	A notification function shall be called to perform notification.

An implementation may define additional notification mechanisms.

The *sigev_signo* member specifies the signal to be generated. The *sigev_value* member is the application-defined value to be passed to the signal-catching function at the time of the signal.

delivery or to be returned at signal acceptance as the *si_value* member of the **siginfo_t** structure.

The **signal** union is defined in **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	<i>sival_int</i>	Integer signal value.
void*	<i>sival_ptr</i>	Pointer signal value.

The *sival_int* member shall be used when the application-defined value is of type **int**; the *sival_ptr* member shall be used when the application-defined value is a pointer.

When a signal is generated by the *sigqueue()* function or any signal-generating function that supports the specification of an application-defined value, the signal shall be marked pending and, if the SA_SIGINFO flag is set for that signal, the signal shall be queued to the process along with the application-specified signal value. Multiple occurrences of signals so generated are queued in FIFO order. It is unspecified whether signals so generated are queued when the SA_SIGINFO flag is not set for that signal.

Signals generated by the *kill()* function or other events that cause signals to occur, such as detection of hardware faults, *alarm()* timer expiration, or terminal activity, and for which the implementation does not support queuing, shall have no effect on signals already queued for the same signal number.

When multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, are pending, the behavior shall be as if the implementation delivers the pending unblocked signal with the lowest signal number within that range. No other ordering of signal delivery is specified.

If, when a pending signal is delivered, there are additional signals queued to that signal number, the signal shall remain pending. Otherwise, the pending indication shall be reset.

Multi-threaded programs can use an alternate event notification mechanism. When a notification is processed, and the *sigev_notify* member of the **sigevent** structure has the value SIGEV_THREAD, the function *sigev_notify_function* is called with parameter *sigev_value*.

The function shall be executed in an environment as if it were the *start_routine* for a newly created thread with thread attributes specified by *sigev_notify_attributes*. If *sigev_notify_attributes* is NULL, the behavior shall be as if the thread were created with the *detachstate* attribute set to PTHREAD_CREATE_DETACHED. Supplying an attributes structure with a *detachstate* attribute of PTHREAD_CREATE_JOINABLE results in undefined behavior. The signal mask of this thread is implementation-defined.

2.4.3 Signal Actions

There are three types of action that can be associated with a signal: SIG_DFL, SIG_IGN, or a pointer to a function. Initially, all signals shall be set to SIG_DFL or SIG_IGN prior to entry of the *main()* routine (see the *exec* functions). The actions prescribed by these values are as follows.

SIG_DFL

Signal-specific default action.

The default actions for the signals defined in this volume of POSIX.1-2008 are specified under **<signal.h>**. The default actions for the realtime signals in the range SIGRTMIN to SIGRTMAX shall be to terminate the process abnormally.

If the default action is to terminate the process abnormally, the process is terminated as if by a call to *_exit()*, except that the status made available to *wait()*, *waitid()*, and *waitpid()* indicates

General Information

Signal Concepts

16628	XSI	abnormal termination by the signal. If the default action is to terminate the process abnormally with additional actions, implementation-defined abnormal termination actions, such as creation of a core file, may also occur.
16629		
16630		
16631		If the default action is to stop the process, the execution of that process is temporarily suspended. When a process stops, a SIGCHLD signal shall be generated for its parent process, unless the parent process has set the SA_NOCLDSTOP flag. While a process is stopped, any additional signals that are sent to the process shall not be delivered until the process is continued, except SIGKILL which always terminates the receiving process. A process that is a member of an orphaned process group shall not be allowed to stop in response to the SIGSTP, SIGTTIN, or SIGTTOU signals. In cases where delivery of one of these signals would stop such a process, the signal shall be discarded.
16632		
16633		
16634		
16635		
16636		
16637		
16638		
16639		If the default action is to ignore the signal, delivery of the signal shall have no effect on the process.
16640		
16641		Setting a signal action to SIG_DFL for a signal that is pending, and whose default action is to ignore the signal (for example, SIGCHLD), shall cause the pending signal to be discarded, whether or not it is blocked. Any queued values pending shall be discarded and the resources used to queue them shall be released and returned to the system for other use.
16642		
16643		
16644		
16645		The default action for SIGCONT is to resume execution at the point where the process was stopped, after first handling any pending unblocked signals.
16646		
16647	XSI	When a stopped process is continued, a SIGCHLD signal may be generated for its parent process, unless the parent process has set the SA_NOCLDSTOP flag.
16648		
16649		SIG_IGN
16650		Ignore signal.
16651		Delivery of the signal shall have no effect on the process. The behavior of a process is undefined after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that was not generated by <i>kill()</i> , <i>sigqueue()</i> , or <i>raise()</i> .
16652		
16653		
16654		The system shall not allow the action for the signals SIGKILL or SIGSTOP to be set to SIG_IGN.
16655		Setting a signal action to SIG_IGN for a signal that is pending shall cause the pending signal to be discarded, whether or not it is blocked.
16656		
16657		If a process sets the action for the SIGCHLD signal to SIG_IGN, the behavior is unspecified, except as specified below.
16658	XSI	
16659		If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling processes shall not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited-for children that were transformed into zombie processes, it shall block until all of its children terminate, and <i>wait()</i> , <i>waitid()</i> , and <i>waitpid()</i> shall fail and set <i>errno</i> to [ECHILD].
16660		
16661		
16662		
16663		
16664		Any queued values pending shall be discarded and the resources used to queue them shall be released and made available to queue other signals.
16665		

Pointer to a Function

Catch signal.

On delivery of the signal, the receiving process is to execute the signal-catching function at the specified address. After returning from the signal-catching function, the receiving process shall resume execution at the point at which it was interrupted.

If the SA_SIGINFO flag for the signal is cleared, the signal-catching function shall be entered as a C-language function call as follows:

```
void func(int signo);
```

If the SA_SIGINFO flag for the signal is set, the signal-catching function shall be entered as a C-language function call as follows:

```
void func(int signo, siginfo_t *info, void *context);
```

where *func* is the specified signal-catching function, *signo* is the signal number of the signal being delivered, and *info* is a pointer to a **siginfo_t** structure defined in **<signal.h>** containing at least the following members:

Member Type	Member Name	Description
int	<i>si_signo</i>	Signal number.
int	<i>si_code</i>	Cause of the signal.
pid_t	<i>si_pid</i>	Sending process ID.
uid_t	<i>si_uid</i>	Real user ID of sending process.
void *	<i>si_addr</i>	Address of faulting instruction.
int	<i>si_status</i>	Exit value or signal.
union sigval	<i>si_value</i>	Signal value.

The *si_signo* member shall contain the signal number. This shall be the same as the *signo* parameter. The *si_code* member shall contain a code identifying the cause of the signal. The following non-signal-specific values are defined for *si_code*:

SI_USER The signal was sent by the *kill()* function. The implementation may set *si_code* to **SI_USER** if the signal was sent by the *raise()* or *abort()* functions or any similar functions provided as implementation extensions.

SI_QUEUE The signal was sent by the *sigqueue()* function.

SI_TIMER The signal was generated by the expiration of a timer set by *timer_settime()*.

SI_ASYNCIO The signal was generated by the completion of an asynchronous I/O request.

MSG SI_MESGQ The signal was generated by the arrival of a message on an empty message queue.

Signal-specific values for *si_code* are also defined, as described in XBD **<signal.h>**.

If the signal was not generated by one of the functions or events listed above, *si_code* shall be set either to one of the signal-specific values described in XBD **<signal.h>**, or to an implementation-defined value that is not equal to any of the values defined above.

XSI If *si_code* is **SI_USER** or **SI_QUEUE**, or any value less than or equal to 0, then the signal was generated by a process and *si_pid* and *si_uid* shall be set to the process ID and the real user ID of the sender, respectively.

In addition, *si_addr*, *si_pid*, *si_status*, and *si_uid* shall be set for certain signal-specific values of *si_code*, as described in XBD **<signal.h>**.

If *si_code* is one of **SI_QUEUE**, **SI_TIMER**, **SI_ASYNCIO**, or **SI_MESGQ**, then *si_value* shall

General Information

Signal Concepts

contain the application-specified signal value. Otherwise, the contents of *si_value* are undefined.

The behavior of a process is undefined after it returns normally from a signal-catching function for a SIGBUS, SIGFPE, SIGILL, or SIGSEGV signal that was not generated by *kill()*, *sigqueue()*, or *raise()*.

The system shall not allow a process to catch the signals SIGKILL and SIGSTOP.

If a process establishes a signal-catching function for the SIGCHLD signal while it has a terminated child process for which it has not waited, it is unspecified whether a SIGCHLD signal is generated to indicate that child process.

When signal-catching functions are invoked asynchronously with process execution, the behavior of some of the functions defined by this volume of POSIX.1-2008 is unspecified if they are called from a signal-catching function.

The following table defines a set of functions that shall be async-signal-safe. Therefore, applications can invoke them, without restriction, from signal-catching functions:

<i>_Exit()</i>	<i>fexecve()</i>	<i>poll()</i>	<i>sigqueue()</i>
<i>_exit()</i>	<i>fork()</i>	<i>posix_trace_event()</i>	<i>sigset()</i>
<i>abort()</i>	<i>fstat()</i>	<i>pselect()</i>	<i>sigsuspend()</i>
<i>accept()</i>	<i>fstatat()</i>	<i>raise()</i>	<i>sleep()</i>
<i>access()</i>	<i>fsync()</i>	<i>read()</i>	<i>socketmark()</i>
<i>aio_error()</i>	<i>ftruncate()</i>	<i>readlink()</i>	<i>socket()</i>
<i>aio_return()</i>	<i>futimens()</i>	<i>readlinkat()</i>	<i>socketpair()</i>
<i>aio_suspend()</i>	<i>getegid()</i>	<i>recv()</i>	<i>stat()</i>
<i>alarm()</i>	<i>geteuid()</i>	<i>recvfrom()</i>	<i>symlink()</i>
<i>bind()</i>	<i>getgid()</i>	<i>recvmsg()</i>	<i>symlinkat()</i>
<i>cfgetispeed()</i>	<i>getgroups()</i>	<i>rename()</i>	<i>tcdrain()</i>
<i>cfgetospeed()</i>	<i>getpeername()</i>	<i>renameat()</i>	<i>tcflow()</i>
<i>cfsetispeed()</i>	<i>getpgrp()</i>	<i>rmdir()</i>	<i>tcflush()</i>
<i>cfsetospeed()</i>	<i>getpid()</i>	<i>select()</i>	<i>tcgetattr()</i>
<i>chdir()</i>	<i>getppid()</i>	<i>sem_post()</i>	<i>tcgetpgrp()</i>
<i>chmod()</i>	<i>getsockname()</i>	<i>send()</i>	<i>tcsendbreak()</i>
<i>chown()</i>	<i>getsockopt()</i>	<i>sendmsg()</i>	<i>tcsetattr()</i>
<i>clock_gettime()</i>	<i>getuid()</i>	<i>sendto()</i>	<i>tcsetpgrp()</i>
<i>close()</i>	<i>kill()</i>	<i>setgid()</i>	<i>time()</i>
<i>connect()</i>	<i>link()</i>	<i>setpgid()</i>	<i>timer_getoverrun()</i>
<i>creat()</i>	<i>linkat()</i>	<i>setsid()</i>	<i>timer_gettime()</i>
<i>dup()</i>	<i>listen()</i>	<i>setsockopt()</i>	<i>timer_settime()</i>
<i>dup2()</i>	<i>lseek()</i>	<i>setuid()</i>	<i>times()</i>
<i>execl()</i>	<i>lstat()</i>	<i>shutdown()</i>	<i>umask()</i>
<i>execle()</i>	<i>mkdir()</i>	<i>sigaction()</i>	<i>uname()</i>
<i>execv()</i>	<i>mkdirat()</i>	<i>sigaddset()</i>	<i>unlink()</i>
<i>execve()</i>	<i>mkfifo()</i>	<i>sigdelset()</i>	<i>unlinkat()</i>
<i>faccessat()</i>	<i>mkfifoat()</i>	<i>sigemptyset()</i>	<i>utime()</i>
<i>fchmod()</i>	<i>mknod()</i>	<i>sigfillset()</i>	<i>utimensat()</i>
<i>fchmodat()</i>	<i>mknodat()</i>	<i>sigismember()</i>	<i>utimes()</i>
<i>fchown()</i>	<i>open()</i>	<i>signal()</i>	<i>wait()</i>
<i>fchownat()</i>	<i>openat()</i>	<i>sigpause()</i>	<i>waitpid()</i>
<i>fcntl()</i>	<i>pause()</i>	<i>sigpending()</i>	<i>write()</i>
<i>fdatasync()</i>	<i>pipe()</i>	<i>sigprocmask()</i>	

All functions not in the above table are considered to be unsafe with respect to signals. In the presence of signals, all functions defined by this volume of POSIX.1-2008 shall behave as defined

16758 when called from or interrupted by a signal-catching function, with a single exception: when a
 16759 signal interrupts an unsafe function and the signal-catching function calls an unsafe function,
 16760 the behavior is undefined.

16761 Operations which obtain the value of *errno* and operations which assign a value to *errno* shall be
 16762 *async-signal-safe*.

16763 When a signal is delivered to a thread, if the action of that signal specifies termination, stop, or
 16764 continue, the entire process shall be terminated, stopped, or continued, respectively.

16765 2.4.4 Signal Effects on Other Functions

16766 Signals affect the behavior of certain functions defined by this volume of POSIX.1-2008 if
 16767 delivered to a process while it is executing such a function. If the action of the signal is to
 16768 terminate the process, the process shall be terminated and the function shall not return. If the
 16769 action of the signal is to stop the process, the process shall stop until continued or terminated.
 16770 Generation of a SIGCONT signal for the process shall cause the process to be continued, and the
 16771 original function shall continue at the point the process was stopped. If the action of the signal is
 16772 to invoke a signal-catching function, the signal-catching function shall be invoked; in this case
 16773 the original function is said to be “interrupted” by the signal. If the signal-catching function
 16774 executes a **return** statement, the behavior of the interrupted function shall be as described
 16775 individually for that function, except as noted for unsafe functions. Signals that are ignored shall
 16776 not affect the behavior of any function; signals that are blocked shall not affect the behavior of
 16777 any function until they are unblocked and then delivered, except as specified for the *sigpending()*
 16778 and *sigwait()* functions.

16779 2.5 Standard I/O Streams

16780 CX A stream is associated with an external file (which may be a physical device) or memory buffer
 16781 CX by “opening” a file or buffer. This may involve “creating” a new file. Creating an existing file
 16782 causes its former contents to be discarded if necessary. If a file can support positioning requests
 16783 (such as a disk file, as opposed to a terminal), then a “file position indicator” associated with the
 16784 stream is positioned at the start (byte number 0) of the file, unless the file is opened with append
 16785 mode, in which case it is implementation-defined whether the file position indicator is initially
 16786 positioned at the beginning or end of the file. The file position indicator is maintained by
 16787 subsequent reads, writes, and positioning requests, to facilitate an orderly progression through
 16788 the file. All input takes place as if bytes were read by successive calls to *fgetc()*; all output takes
 16789 place as if bytes were written by successive calls to *fputc()*.

16790 When a stream is “unbuffered”, bytes are intended to appear from the source or at the
 16791 destination as soon as possible; otherwise, bytes may be accumulated and transmitted as a
 16792 block. When a stream is “fully buffered”, bytes are intended to be transmitted as a block when a
 16793 buffer is filled. When a stream is “line buffered”, bytes are intended to be transmitted as a block
 16794 when a <newline> byte is encountered. Furthermore, bytes are intended to be transmitted as a
 16795 block when a buffer is filled, when input is requested on an unbuffered stream, or when input is
 16796 requested on a line-buffered stream that requires the transmission of bytes. Support for these
 16797 characteristics is implementation-defined, and may be affected via *setbuf()* and *setvbuf()*.

16798 A file may be disassociated from a controlling stream by “closing” the file. Output streams are
 16799 flushed (any unwritten buffer contents are transmitted) before the stream is disassociated from
 16800 the file. The value of a pointer to a **FILE** object is unspecified after the associated file is closed
 16801 (including the standard streams).

A file may be subsequently reopened, by the same or another program execution, and its contents reclaimed or modified (if it can be repositioned at its start). If the *main()* function returns to its original caller, or if the *exit()* function is called, all open files are closed (hence all output streams are flushed) before program termination. Other paths to program termination, such as calling *abort()*, need not close all files properly.

The address of the **FILE** object used to control a stream may be significant; a copy of a **FILE** object need not necessarily serve in place of the original.

At program start-up, three streams are predefined and need not be opened explicitly: *standard input* (for reading conventional input), *standard output* (for writing conventional output), and *standard error* (for writing diagnostic output). When opened, the standard error stream is not fully buffered; the standard input and standard output streams are fully buffered if and only if the stream can be determined not to refer to an interactive device.

A stream associated with a memory buffer shall have the same operations for text files that a stream associated with an external file would have. In addition, the stream orientation shall be determined in exactly the same fashion.

Input and output operations on a stream associated with a memory buffer by a call to *fmemopen()* shall be constrained by the implementation to take place within the bounds of the memory buffer. In the case of a stream opened by *open_memstream()* or *open_wmemstream()*, the memory area shall grow dynamically to accommodate write operations as necessary. For output, data is moved from the buffer provided by *setvbuf()* to the memory stream during a flush or close operation.

2.5.1 Interaction of File Descriptors and Standard I/O Streams

This section describes the interaction of file descriptors and standard I/O streams. The functionality described in this section is an extension to the ISO C standard (and the rest of this section is not further CX shaded).

An open file description may be accessed through a file descriptor, which is created using functions such as *open()* or *pipe()*, or through a stream, which is created using functions such as *fopen()* or *popen()*. Either a file descriptor or a stream is called a “handle” on the open file description to which it refers; an open file description may have several handles.

Handles can be created or destroyed by explicit user action, without affecting the underlying open file description. Some of the ways to create them include *fcntl()*, *dup()*, *fdopen()*, *fileno()*, and *fork()*. They can be destroyed by at least *fclose()*, *close()*, and the *exec* functions.

A file descriptor that is never used in an operation that could affect the file offset (for example, *read()*, *write()*, or *lseek()*) is not considered a handle for this discussion, but could give rise to one (for example, as a consequence of *fdopen()*, *dup()*, or *fork()*). This exception does not include the file descriptor underlying a stream, whether created with *fopen()* or *fdopen()*, so long as it is not used directly by the application to affect the file offset. The *read()* and *write()* functions implicitly affect the file offset; *lseek()* explicitly affects it.

The result of function calls involving any one handle (the “active handle”) is defined elsewhere in this volume of POSIX.1-2008, but if two or more handles are used, and any one of them is a stream, the application shall ensure that their actions are coordinated as described below. If this is not done, the result is undefined.

A handle which is a stream is considered to be closed when either an *fclose()* or *freopen()* is executed on it (the result of *freopen()* is a new stream, which cannot be a handle on the same open file description as its previous value), or when the process owning that stream terminates with *exit()*, *abort()*, or due to a signal. A file descriptor is closed by *close()*, *_exit()*, or the *exec*

functions when `FD_CLOEXEC` is set on that file descriptor.

For a handle to become the active handle, the application shall ensure that the actions below are performed between the last use of the handle (the current active handle) and the first use of the second handle (the future active handle). The second handle then becomes the active handle. All activity by the application affecting the file offset on the first handle shall be suspended until it again becomes the active file handle. (If a stream function has as an underlying function one that affects the file offset, the stream function shall be considered to affect the file offset.)

The handles need not be in the same process for these rules to apply.

Note that after a `fork()`, two handles exist where one existed before. The application shall ensure that, if both handles can ever be accessed, they are both in a state where the other could become the active handle first. The application shall prepare for a `fork()` exactly as if it were a change of active handle. (If the only action performed by one of the processes is one of the `exec` functions or `_exit()` (not `exit()`), the handle is never accessed in that process.)

For the first handle, the first applicable condition below applies. After the actions required below are taken, if the handle is still open, the application can close it.

- If it is a file descriptor, no action is required.
- If the only further action to be performed on any handle to this open file descriptor is to close it, no action need be taken.
- If it is a stream which is unbuffered, no action need be taken.
- If it is a stream which is line buffered, and the last byte written to the stream was a `<newline>` (that is, as if a:

```
putc( '\n' )
```

was the most recent operation on that stream), no action need be taken.

- If it is a stream which is open for writing or appending (but not also open for reading), the application shall either perform an `fflush()`, or the stream shall be closed.
- If the stream is open for reading and it is at the end of the file (`feof()` is true), no action need be taken.
- If the stream is open with a mode that allows reading and the underlying open file description refers to a device that is capable of seeking, the application shall either perform an `fflush()` or the stream shall be closed.

Otherwise, the result is undefined.

For the second handle:

- If any previous active handle has been used by a function that explicitly changed the file offset, except as required above for the first handle, the application shall perform an `lseek()` or `fseek()` (as appropriate to the type of handle) to an appropriate location.

If the active handle ceases to be accessible before the requirements on the first handle, above, have been met, the state of the open file description becomes undefined. This might occur during functions such as a `fork()` or `_exit()`.

The `exec` functions make inaccessible all streams that are open at the time they are called, independent of which streams or file descriptors may be available to the new process image.

When these rules are followed, regardless of the sequence of handles used, implementations shall ensure that an application, even one consisting of several processes, shall yield correct

results: no data shall be lost or duplicated when writing, and all data shall be written in order, except as requested by seeks. It is implementation-defined whether, and under what conditions, all input is seen exactly once.

If the rules above are not followed, the result is unspecified.

Each function that operates on a stream is said to have zero or more “underlying functions”. This means that the stream function shares certain traits with the underlying functions, but does not require that there be any relation between the implementations of the stream function and its underlying functions.

2.5.2 Stream Orientation and Encoding Rules

For conformance to the ISO/IEC 9899:1999 standard, the definition of a stream includes an “orientation”. After a stream is associated with an external file, but before any operations are performed on it, the stream is without orientation. Once a wide-character input/output function has been applied to a stream without orientation, the stream shall become “wide-oriented”. Similarly, once a byte input/output function has been applied to a stream without orientation, the stream shall become “byte-oriented”. Only a call to the *freopen()* function or the *fwide()* function can otherwise alter the orientation of a stream.

A successful call to *freopen()* shall remove any orientation. The three predefined streams *standard input*, *standard output*, and *standard error* shall be unoriented at program start-up.

Byte input/output functions cannot be applied to a wide-oriented stream, and wide-character input/output functions cannot be applied to a byte-oriented stream. The remaining stream operations shall not affect and shall not be affected by a stream’s orientation, except for the following additional restriction:

- For wide-oriented streams, after a successful call to a file-positioning function that leaves the file position indicator prior to the end-of-file, a wide-character output function can overwrite a partial character; any file contents beyond the byte(s) written are henceforth undefined.

Each wide-oriented stream has an associated **mbstate_t** object that stores the current parse state of the stream. A successful call to *fgetpos()* shall store a representation of the value of this **mbstate_t** object as part of the value of the **fpos_t** object. A later successful call to *fsetpos()* using the same stored **fpos_t** value shall restore the value of the associated **mbstate_t** object as well as the position within the controlled stream.

Implementations that support multiple encoding rules associate an encoding rule with the stream. The encoding rule shall be determined by the setting of the *LC_CTYPE* category in the current locale at the time when the stream becomes wide-oriented. As with the stream’s orientation, the encoding rule associated with a stream cannot be changed once it has been set, except by a successful call to *freopen()* which clears the encoding rule and resets the orientation to unoriented.

Although wide-oriented streams are conceptually sequences of wide characters, the external file associated with a wide-oriented stream is a sequence of (possibly multi-byte) characters generalized as follows:

- Multi-byte encodings within files may contain embedded null bytes (unlike multi-byte encodings valid for use internal to the program).
- A file need not begin nor end in the initial shift state.

Moreover, the encodings used for characters may differ among files. Both the nature and choice of such encodings are implementation-defined.

16935 The wide-character input functions read characters from the stream and convert them to wide
 16936 characters as if they were read by successive calls to the *fgetcwc()* function. Each conversion shall
 16937 occur as if by a call to the *mbrtowc()* function, with the conversion state described by the
 16938 CX stream's own **mbstate_t** object, except the encoding rule associated with the stream is used
 16939 instead of the encoding rule implied by the *LC_CTYPE* category of the current locale.

16940 The wide-character output functions convert wide characters to (possibly multi-byte) characters
 16941 and write them to the stream as if they were written by successive calls to the *fputcwc()* function.
 16942 Each conversion shall occur as if by a call to the *wcrtomb()* function, with the conversion state
 16943 CX described by the stream's own **mbstate_t** object, except the encoding rule associated with the
 16944 stream is used instead of the encoding rule implied by the *LC_CTYPE* category of the current
 16945 locale.

16946 An "encoding error" shall occur if the character sequence presented to the underlying *mbrtowc()*
 16947 function does not form a valid (generalized) character, or if the code value passed to the
 16948 underlying *wcrtomb()* function does not correspond to a valid (generalized) character. The wide-
 16949 character input/output functions and the byte input/output functions store the value of the
 16950 macro [EILSEQ] in *errno* if and only if an encoding error occurs.

16951 2.6 STREAMS

16952 OB XSR STREAMS functionality is provided on implementations supporting the XSI STREAMS Option
 16953 Group. The functionality described in this section is dependent on support of the XSI STREAMS
 16954 option (and the rest of this section is not further shaded for this option).

16955 STREAMS provides a uniform mechanism for implementing networking services and other
 16956 character-based I/O. The STREAMS function provides direct access to protocol modules.
 16957 STREAMS modules are unspecified objects. Access to STREAMS modules is provided by
 16958 interfaces in POSIX.1-2008. Creation of STREAMS modules is outside the scope of
 16959 POSIX.1-2008.

16960 A STREAM is typically a full-duplex connection between a process and an open device or
 16961 pseudo-device. However, since pipes may be STREAMS-based, a STREAM can be a full-duplex
 16962 connection between two processes. The STREAM itself exists entirely within the implementation
 16963 and provides a general character I/O function for processes. It optionally includes one or more
 16964 intermediate processing modules that are interposed between the process end of the STREAM
 16965 (STREAM head) and a device driver at the end of the STREAM (STREAM end).

16966 STREAMS I/O is based on messages. There are three types of message:

- 16967 • *Data messages* containing actual data for input or output
- 16968 • *Control data* containing instructions for the STREAMS modules and underlying
 16969 implementation
- 16970 • Other messages, which include file descriptors

16971 The interface between the STREAM and the rest of the implementation is provided by a set of
 16972 functions at the STREAM head. When a process calls *write()*, *writew()*, *putmsg()*, *putpmsg()*, or
 16973 *ioctl()*, messages are sent down the STREAM, and *read()*, *readw()*, *getmsg()*, or *getpmsg()* accepts
 16974 data from the STREAM and passes it to a process. Data intended for the device at the
 16975 downstream end of the STREAM is packaged into messages and sent downstream, while data
 16976 and signals from the device are composed into messages by the device driver and sent upstream
 16977 to the STREAM head.

16978 When a STREAMS-based device is opened, a STREAM shall be created that contains the

STREAM head and the STREAM end (driver). If pipes are STREAMS-based in an implementation, when a pipe is created, two STREAMS shall be created, each containing a STREAM head. Other modules are added to the STREAM using *ioctl()*. New modules are “pushed” onto the STREAM one at a time in last-in, first-out (LIFO) style, as though the STREAM was a push-down stack.

Priority

Message types are classified according to their queuing priority and may be *normal* (non-priority), *priority*, or *high-priority* messages. A message belongs to a particular priority band that determines its ordering when placed on a queue. Normal messages have a priority band of 0 and shall always be placed at the end of the queue following all other messages in the queue. High-priority messages are always placed at the head of a queue, but shall be discarded if there is already a high-priority message in the queue. Their priority band shall be ignored; they are high-priority by virtue of their type. Priority messages have a priority band greater than 0. Priority messages are always placed after any messages of the same or higher priority. High-priority and priority messages are used to send control and data information outside the normal flow of control. By convention, high-priority messages shall not be affected by flow control. Normal and priority messages have separate flow controls.

Message Parts

A process may access STREAMS messages that contain a data part, control part, or both. The data part is that information which is transmitted over the communication medium and the control information is used by the local STREAMS modules. The other types of messages are used between modules and are not accessible to processes. Messages containing only a data part are accessible via *putmsg()*, *putpmsg()*, *getmsg()*, *getpmsg()*, *read()*, *readv()*, *write()*, or *writv()*. Messages containing a control part with or without a data part are accessible via calls to *putmsg()*, *putpmsg()*, *getmsg()*, or *getpmsg()*.

2.6.1 Accessing STREAMS

A process accesses STREAMS-based files using the standard functions *close()*, *ioctl()*, *getmsg()*, *getpmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *putpmsg()*, *read()*, or *write()*. Refer to the applicable function definitions for general properties and errors.

Calls to *ioctl()* shall perform control functions on the STREAM associated with the file descriptor *fd*. The control functions may be performed by the STREAM head, a STREAMS module, or the STREAMS driver for the STREAM.

STREAMS modules and drivers can detect errors, sending an error message to the STREAM head, thus causing subsequent functions to fail and set *errno* to the value specified in the message. In addition, STREAMS modules and drivers can elect to fail a particular *ioctl()* request alone by sending a negative acknowledgement message to the STREAM head. This shall cause just the pending *ioctl()* request to fail and set *errno* to the value specified in the message.

2.7 XSI Interprocess Communication

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

The following message passing, semaphore, and shared memory services form an XSI interprocess communication facility. Certain aspects of their operation are common, and are defined as follows.

IPC Functions		
<i>msgctl()</i>	<i>semctl()</i>	<i>shmat()</i>
<i>msgget()</i>	<i>semget()</i>	<i>shmctl()</i>
<i>msgrcv()</i>	<i>semop()</i>	<i>shmdt()</i>
<i>msgsnd()</i>		<i>shmget()</i>

Another interprocess communication facility is provided by functions in the Realtime Option Group; see [Section 2.8](#) (on page 497).

2.7.1 IPC General Description

Each individual shared memory segment, message queue, and semaphore set shall be identified by a unique positive integer, called, respectively, a shared memory identifier, *shmid*, a semaphore identifier, *semid*, and a message queue identifier, *msgid*. The identifiers shall be returned by calls to *shmget()*, *semget()*, and *msgget()*, respectively.

Associated with each identifier is a data structure which contains data related to the operations which may be or may have been performed; see the Base Definitions volume of POSIX.1-2008, **<sys/shm.h>**, **<sys/sem.h>**, and **<sys/msg.h>** for their descriptions.

Each of the data structures contains both ownership information and an **ipc_perm** structure (see the Base Definitions volume of POSIX.1-2008, **<sys/ipc.h>**) which are used in conjunction to determine whether or not read/write (read/alter for semaphores) permissions should be granted to processes using the IPC facilities. The *mode* member of the **ipc_perm** structure acts as a bit field which determines the permissions.

The values of the bits are given below in octal notation.

Bit	Meaning
0400	Read by user.
0200	Write by user.
0040	Read by group.
0020	Write by group.
0004	Read by others.
0002	Write by others.

The name of the **ipc_perm** structure is *shm_perm*, *sem_perm*, or *msg_perm*, depending on which service is being used. In each case, read and write/alter permissions shall be granted to a process if one or more of the following are true ("xxx" is replaced by *shm*, *sem*, or *msg*, as appropriate):

- The process has appropriate privileges.
- The effective user ID of the process matches *xxx_perm.cuid* or *xxx_perm.uid* in the data structure associated with the IPC identifier, and the appropriate bit of the *user* field in *xxx_perm.mode* is set.

General Information

XSI Interprocess Communication

- The effective user ID of the process does not match `xxx_perm.cuid` or `xxx_perm.uid` but the effective group ID of the process matches `xxx_perm.cgid` or `xxx_perm.gid` in the data structure associated with the IPC identifier, and the appropriate bit of the `group` field in `xxx_perm.mode` is set.
- The effective user ID of the process does not match `xxx_perm.cuid` or `xxx_perm.uid` and the effective group ID of the process does not match `xxx_perm.cgid` or `xxx_perm.gid` in the data structure associated with the IPC identifier, but the appropriate bit of the `other` field in `xxx_perm.mode` is set.

Otherwise, the permission shall be denied.

2.8 Realtime

This section defines functions to support the source portability of applications with realtime requirements. The presence of some of these functions is dependent on support for implementation options described in the text.

The specific functional areas included in this section and their scope include the following. Full definitions of these terms can be found in XBD [Chapter 3](#) (on page 33).

- Semaphores
- Process Memory Locking
- Memory Mapped Files and Shared Memory Objects
- Priority Scheduling
- Realtime Signal Extension
- Timers
- Interprocess Communication
- Synchronized Input and Output
- Asynchronous Input and Output

All the realtime functions defined in this volume of POSIX.1-2008 are portable, although some of the numeric parameters used by an implementation may have hardware dependencies.

2.8.1 Realtime Signals

See [Section 2.4.2](#) (on page 485).

2.8.2 Asynchronous I/O

An asynchronous I/O control block structure `aiocb` is used in many asynchronous I/O functions. It is defined in the Base Definitions volume of POSIX.1-2008, `<aio.h>` and has at least the following members:

Member Type	Member Name	Description
int	<i>aio_fildes</i>	File descriptor.
off_t	<i>aio_offset</i>	File offset.
volatile void*	<i>aio_buf</i>	Location of buffer.
size_t	<i>aio_nbytes</i>	Length of transfer.
int	<i>aio_reqprio</i>	Request priority offset.
struct sigevent	<i>aio_sigtvent</i>	Signal number and value.
int	<i>aio_lio_opcode</i>	Operation to be performed.

The *aio_fildes* element is the file descriptor on which the asynchronous operation is performed.

If `O_APPEND` is not set for the file descriptor *aio_fildes* and if *aio_fildes* is associated with a device that is capable of seeking, then the requested operation takes place at the absolute position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to the operation with an *offset* argument equal to *aio_offset* and a *whence* argument equal to `SEEK_SET`. If `O_APPEND` is set for the file descriptor, or if *aio_fildes* is associated with a device that is incapable of seeking, write operations append to the file in the same order as the calls were made, with the following exception: under implementation-defined circumstances, such as operation on a multi-processor or when requests of differing priorities are submitted at the same time, the ordering restriction may be relaxed. Since there is no way for a strictly conforming application to determine whether this relaxation applies, all strictly conforming applications which rely on ordering of output shall be written in such a way that they will operate correctly if the relaxation applies. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified. The *aio_nbytes* and *aio_buf* elements are the same as the *nbyte* and *buf* arguments defined by *read()* and *write()*, respectively.

If `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` are defined, then asynchronous I/O is queued in priority order, with the priority of each asynchronous operation based on the current scheduling priority of the calling process. The *aio_reqprio* member can be used to lower (but not raise) the asynchronous I/O operation priority and is within the range zero through `[AIO_PRIO_DELTA_MAX]`, inclusive. Unless both `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing asynchronous I/O requests is unspecified. When both `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing of requests submitted by processes whose schedulers are not `SCHED_FIFO`, `SCHED_RR`, or `SCHED_SPORADIC` is unspecified. The priority of an asynchronous request is computed as (process scheduling priority) minus *aio_reqprio*. The priority assigned to each asynchronous I/O request is an indication of the desired order of execution of the request relative to other asynchronous I/O requests for this file. If `_POSIX_PRIORITIZED_IO` is defined, requests issued with the same priority to a character special file are processed by the underlying device in FIFO order; the order of processing of requests of the same priority issued to files that are not character special files is unspecified. Numerically higher priority values indicate requests of higher priority. The value of *aio_reqprio* has no effect on process scheduling priority. When prioritized asynchronous I/O requests to the same file are blocked waiting for a resource required for that I/O operation, the higher-priority I/O requests shall be granted the resource before lower-priority I/O requests are granted the resource. The relative priority of asynchronous I/O and synchronous I/O is implementation-defined. If `_POSIX_PRIORITIZED_IO` is defined, the implementation shall define for which files I/O prioritization is supported.

The *aio_sigtvent* determines how the calling process shall be notified upon I/O completion, as specified in [Section 2.4.1](#) (on page 484). If *aio_sigtvent.sigev_notify* is `SIGEV_NONE`, then no signal shall be posted upon I/O completion, but the error status for the operation and the return status for the operation shall be set appropriately.

The *aio_lio_opcode* field is used only by the *lio_listio()* call. The *lio_listio()* call allows multiple

asynchronous I/O operations to be submitted at a single time. The function takes as an argument an array of pointers to **aio** structures. Each **aio** structure indicates the operation to be performed (read or write) via the *aio_lio_opcode* field.

The address of the **aio** structure is used as a handle for retrieving the error status and return status of the asynchronous operation while it is in progress.

The **aio** structure and the data buffers associated with the asynchronous I/O operation are being used by the system for asynchronous I/O while, and only while, the error status of the asynchronous operation is equal to [EINPROGRESS]. Applications shall not modify the **aio** structure while the structure is being used by the system for asynchronous I/O.

The return status of the asynchronous operation is the number of bytes transferred by the I/O operation. If the error status is set to indicate an error completion, then the return status is set to the return value that the corresponding *read()*, *write()*, or *fsync()* call would have returned. When the error status is not equal to [EINPROGRESS], the return status shall reflect the return status of the corresponding synchronous operation.

2.8.3 Memory Management

2.8.3.1 Memory Locking

Range memory locking operations are defined in terms of pages. Implementations may restrict the size and alignment of range lockings to be on page-size boundaries. The page size, in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has no restrictions on size or alignment, it may specify a 1-byte page size.

Memory locking guarantees the residence of portions of the address space. It is implementation-defined whether locking memory guarantees fixed translation between virtual addresses (as seen by the process) and physical addresses. Per-process memory locks are not inherited across a *fork()*, and all memory locks owned by a process are unlocked upon *exec* or process termination. Unmapping of an address range removes any memory locks established on that address range by this process.

2.8.3.2 Memory Mapped Files

Range memory mapping operations are defined in terms of pages. Implementations may restrict the size and alignment of range mappings to be on page-size boundaries. The page size, in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has no restrictions on size or alignment, it may specify a 1-byte page size.

Memory mapped files provide a mechanism that allows a process to access files by directly incorporating file data into its address space. Once a file is mapped into a process address space, the data can be manipulated as memory. If more than one process maps a file, its contents are shared among them. If the mappings allow shared write access, then data written into the memory object through the address space of one process appears in the address spaces of all processes that similarly map the same portion of the memory object.

Shared memory objects are named regions of storage that may be independent of the file system and can be mapped into the address space of one or more processes to allow them to share the associated memory.

17181 SHM An *unlink()* of a file or *shm_unlink()* of a shared memory object, while causing the removal of
 17182 the name, does not unmap any mappings established for the object. Once the name has been
 17183 removed, the contents of the memory object are preserved as long as it is referenced. The
 17184 memory object remains referenced as long as a process has the memory object open or has some
 17185 area of the memory object mapped.

17186 2.8.3.3 Memory Protection

17187 When an object is mapped, various application accesses to the mapped region may result in
 17188 signals. In this context, SIGBUS is used to indicate an error using the mapped object, and
 17189 SIGSEGV is used to indicate a protection violation or misuse of an address:

- 17190 • A mapping may be restricted to disallow some types of access.
- 17191 • Write attempts to memory that was mapped without write access, or any access to
 17192 memory mapped PROT_NONE, shall result in a SIGSEGV signal.
- 17193 • References to unmapped addresses shall result in a SIGSEGV signal.
- 17194 • Reference to whole pages within the mapping, but beyond the current length of the object,
 17195 shall result in a SIGBUS signal.
- 17196 • The size of the object is unaffected by access beyond the end of the object (even if a
 17197 SIGBUS is not generated).

17198 2.8.3.4 Typed Memory Objects

17199 TYM The functionality described in this section shall be provided on implementations that support
 17200 the Typed Memory Objects option (and the rest of this section is not further shaded for this
 17201 option).

17202 Implementations may support the Typed Memory Objects option independently of support for
 17203 memory mapped files or shared memory objects. Typed memory objects are implementation-
 17204 configurable named storage pools accessible from one or more processors in a system, each via
 17205 one or more ports, such as backplane buses, LANs, I/O channels, and so on. Each valid
 17206 combination of a storage pool and a port is identified through a name that is defined at system
 17207 configuration time, in an implementation-defined manner; the name may be independent of the
 17208 file system. Using this name, a typed memory object can be opened and mapped into process
 17209 address space. For a given storage pool and port, it is necessary to support both dynamic
 17210 allocation from the pool as well as mapping at an application-supplied offset within the pool;
 17211 when dynamic allocation has been performed, subsequent deallocation must be supported.
 17212 Lastly, accessing typed memory objects from different ports requires a method for obtaining the
 17213 offset and length of contiguous storage of a region of typed memory (dynamically allocated or
 17214 not); this allows typed memory to be shared among processes and/or processors while being
 17215 accessed from the desired port.

2.8.4 Process Scheduling

PS The functionality described in this section shall be provided on implementations that support the Process Scheduling option (and the rest of this section is not further shaded for this option).

Scheduling Policies

The scheduling semantics described in this volume of POSIX.1-2008 are defined in terms of a conceptual model that contains a set of thread lists. No implementation structures are necessarily implied by the use of this conceptual model. It is assumed that no time elapses during operations described using this model, and therefore no simultaneous operations are possible. This model discusses only processor scheduling for runnable threads, but it should be noted that greatly enhanced predictability of realtime applications results if the sequencing of other resources takes processor scheduling policy into account.

There is, conceptually, one thread list for each priority. A runnable thread will be on the thread list for that thread's priority. Multiple scheduling policies shall be provided. Each non-empty thread list is ordered, contains a head as one end of its order, and a tail as the other. The purpose of a scheduling policy is to define the allowable operations on this set of lists (for example, moving threads between and within lists).

The POSIX model treats a "process" as an aggregation of system resources, including one or more threads that may be scheduled by the operating system on the processor(s) it controls. Although a process has its own set of scheduling attributes, these have an indirect effect (if any) on the scheduling behavior of individual threads as described below.

Each thread shall be controlled by an associated scheduling policy and priority. These parameters may be specified by explicit application execution of the `pthread_setschedparam()` function. Additionally, the scheduling parameters of a thread (but not its scheduling policy) may be changed by application execution of the `pthread_setschedprio()` function.

Each process shall be controlled by an associated scheduling policy and priority. These parameters may be specified by explicit application execution of the `sched_setscheduler()` or `sched_setparam()` functions.

The effect of the process scheduling attributes on individual threads in the process is dependent on the scheduling contention scope of the threads (see [Section 2.9.4](#), on page 509):

- For threads with system scheduling contention scope, the process scheduling attributes shall have no effect on the scheduling attributes or behavior either of the thread or an underlying kernel scheduling entity dedicated to that thread.
- For threads with process scheduling contention scope, the process scheduling attributes shall have no effect on the scheduling attributes of the thread. However, any underlying kernel scheduling entity used by these threads shall at all times behave as specified by the scheduling attributes of the containing process, and this behavior may affect the scheduling behavior of the process contention scope threads. For example, a process contention scope thread with scheduling policy `SCHED_FIFO` and the system maximum priority H (the value returned by `sched_get_priority_max(SCHED_FIFO)`) in a process with scheduling policy `SCHED_RR` and system minimum priority L (the value returned by `sched_get_priority_min(SCHED_RR)`) shall be subject to timeslicing and to preemption by any thread with an effective priority higher than L .

Associated with each policy is a priority range. Each policy definition shall specify the minimum priority range for that policy. The priority ranges for each policy may but need not overlap the priority ranges of other policies.

A conforming implementation shall select the thread that is defined as being at the head of the

- 17262 highest priority non-empty thread list to become a running thread, regardless of its associated
17263 policy. This thread is then removed from its thread list.
- 17264 Four scheduling policies are specifically required. Other implementation-defined scheduling
17265 policies may be defined. The following symbols are defined in the Base Definitions volume of
17266 POSIX.1-2008, <**sched.h**>:
- 17267 **SCHED_FIFO** First in, first out (FIFO) scheduling policy.
- 17268 **SCHED_RR** Round robin scheduling policy.
- 17269 **SS** **SCHED_SPORADIC** Sporadic server scheduling policy.
- 17270 **SCHED_OTHER** Another scheduling policy.
- 17271 The values of these symbols shall be distinct.
- 17272 **SCHED_FIFO**
- 17273 Conforming implementations shall include a scheduling policy called the FIFO scheduling
17274 policy.
- 17275 Threads scheduled under this policy are chosen from a thread list that is ordered by the time its
17276 threads have been on the list without being executed; generally, the head of the list is the thread
17277 that has been on the list the longest time, and the tail is the thread that has been on the list the
17278 shortest time.
- 17279 Under the **SCHED_FIFO** policy, the modification of the definitional thread lists is as follows:
- 17280 1. When a running thread becomes a preempted thread, it becomes the head of the thread
17281 list for its priority.
 - 17282 2. When a blocked thread becomes a runnable thread, it becomes the tail of the thread list
17283 for its priority.
 - 17284 3. When a running thread calls the *sched_setscheduler()* function, the process specified in the
17285 function call is modified to the specified policy and the priority specified by the *param*
17286 argument.
 - 17287 4. When a running thread calls the *sched_setparam()* function, the priority of the process
17288 specified in the function call is modified to the priority specified by the *param* argument.
 - 17289 5. When a running thread calls the *pthread_setschedparam()* function, the thread specified in
17290 the function call is modified to the specified policy and the priority specified by the *param*
17291 argument.
 - 17292 6. When a running thread calls the *pthread_setschedprio()* function, the thread specified in the
17293 function call is modified to the priority specified by the *prio* argument.
 - 17294 7. If a thread whose policy or priority has been modified other than by *pthread_setschedprio()*
17295 is a running thread or is runnable, it then becomes the tail of the thread list for its new
17296 priority.
 - 17297 8. If a thread whose priority has been modified by *pthread_setschedprio()* is a running thread
17298 or is runnable, the effect on its position in the thread list depends on the direction of the
17299 modification, as follows:
 - 17300 a. If the priority is raised, the thread becomes the tail of the thread list.
 - 17301 b. If the priority is unchanged, the thread does not change position in the thread list.

c. If the priority is lowered, the thread becomes the head of the thread list.

9. When a running thread issues the *sched_yield()* function, the thread becomes the tail of the thread list for its priority.

10. At no other time is the position of a thread with this scheduling policy within the thread lists affected.

For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max()* and *sched_get_priority_min()* functions when SCHED_FIFO is provided as the parameter. Conforming implementations shall provide a priority range of at least 32 priorities for this policy.

SCHED_RR

Conforming implementations shall include a scheduling policy called the “round robin” scheduling policy. This policy shall be identical to the SCHED_FIFO policy with the additional condition that when the implementation detects that a running thread has been executing as a running thread for a time period of the length returned by the *sched_rr_get_interval()* function or longer, the thread shall become the tail of its thread list and the head of that thread list shall be removed and made a running thread.

The effect of this policy is to ensure that if there are multiple SCHED_RR threads at the same priority, one of them does not monopolize the processor. An application should not rely only on the use of SCHED_RR to ensure application progress among multiple threads if the application includes threads using the SCHED_FIFO policy at the same or higher priority levels or SCHED_RR threads at a higher priority level.

A thread under this policy that is preempted and subsequently resumes execution as a running thread completes the unexpired portion of its round robin interval time period.

For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max()* and *sched_get_priority_min()* functions when SCHED_RR is provided as the parameter. Conforming implementations shall provide a priority range of at least 32 priorities for this policy.

SCHED_SPORADIC

The functionality described in this section shall be provided on implementations that support the Process Sporadic Server or Thread Sporadic Server options (and the rest of this section is not further shaded for these options).

If `_POSIX_SPORADIC_SERVER` or `_POSIX_THREAD_SPORADIC_SERVER` is defined, the implementation shall include a scheduling policy identified by the value SCHED_SPORADIC.

The sporadic server policy is based primarily on two time parameters: the replenishment period and the available execution capacity. The replenishment period is given by the *sched_ss_repl_period* member of the **sched_param** structure. The available execution capacity is initialized to the value given by the *sched_ss_init_budget* member of the same parameter. The sporadic server policy is identical to the SCHED_FIFO policy with some additional conditions that cause the thread’s assigned priority to be switched between the values specified by the *sched_priority* and *sched_ss_low_priority* members of the **sched_param** structure.

The priority assigned to a thread using the sporadic server scheduling policy is determined in the following manner: if the available execution capacity is greater than zero and the number of pending replenishment operations is strictly less than *sched_ss_max_repl*, the thread is assigned the priority specified by *sched_priority*; otherwise, the assigned priority shall be *sched_ss_low_priority*. If the value of *sched_priority* is less than or equal to the value of

sched_ss_low_priority, the results are undefined. When active, the thread shall belong to the thread list corresponding to its assigned priority level, according to the mentioned priority assignment. The modification of the available execution capacity and, consequently of the assigned priority, is done as follows:

1. When the thread at the head of the *sched_priority* list becomes a running thread, its execution time shall be limited to at most its available execution capacity, plus the resolution of the execution time clock used for this scheduling policy. This resolution shall be implementation-defined.
2. Each time the thread is inserted at the tail of the list associated with *sched_priority*—because as a blocked thread it became runnable with priority *sched_priority* or because a replenishment operation was performed—the time at which this operation is done is posted as the *activation_time*.
3. When the running thread with assigned priority equal to *sched_priority* becomes a preempted thread, it becomes the head of the thread list for its priority, and the execution time consumed is subtracted from the available execution capacity. If the available execution capacity would become negative by this operation, it shall be set to zero.
4. When the running thread with assigned priority equal to *sched_priority* becomes a blocked thread, the execution time consumed is subtracted from the available execution capacity, and a replenishment operation is scheduled, as described in 6 and 7. If the available execution capacity would become negative by this operation, it shall be set to zero.
5. When the running thread with assigned priority equal to *sched_priority* reaches the limit imposed on its execution time, it becomes the tail of the thread list for *sched_ss_low_priority*, the execution time consumed is subtracted from the available execution capacity (which becomes zero), and a replenishment operation is scheduled, as described in 6 and 7.
6. Each time a replenishment operation is scheduled, the amount of execution capacity to be replenished, *replenish_amount*, is set equal to the execution time consumed by the thread since the *activation_time*. The replenishment is scheduled to occur at *activation_time* plus *sched_ss_repl_period*. If the scheduled time obtained is before the current time, the replenishment operation is carried out immediately. Several replenishment operations may be pending at the same time, each of which will be serviced at its respective scheduled time. With the above rules, the number of replenishment operations simultaneously pending for a given thread that is scheduled under the sporadic server policy shall not be greater than *sched_ss_max_repl*.
7. A replenishment operation consists of adding the corresponding *replenish_amount* to the available execution capacity at the scheduled time. If, as a consequence of this operation, the execution capacity would become larger than *sched_ss_initial_budget*, it shall be rounded down to a value equal to *sched_ss_initial_budget*. Additionally, if the thread was runnable or running, and had assigned priority equal to *sched_ss_low_priority*, then it becomes the tail of the thread list for *sched_priority*.

Execution time is defined in XBD [Section 3.118](#) (on page 52).

For this policy, changing the value of a CPU-time clock via *clock_settime()* shall have no effect on its behavior.

For this policy, valid priorities shall be within the range returned by the *sched_get_priority_min()* and *sched_get_priority_max()* functions when SCHED_SPORADIC is provided as the parameter. Conforming implementations shall provide a priority range of at least 32 distinct priorities for this policy.

If the scheduling policy of the target process is either SCHED_FIFO or SCHED_RR, the *sched_ss_low_priority*, *sched_ss_repl_period*, and *sched_ss_init* budget members of the *param* argument shall have no effect on the scheduling behavior. If the scheduling policy of this process is not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC, the effects of these members are implementation-defined; this case includes the SCHED_OTHER policy.

SCHED_OTHER

Conforming implementations shall include one scheduling policy identified as SCHED_OTHER (which may execute identically with either the FIFO or round robin scheduling policy). The effect of scheduling threads with the SCHED_OTHER policy in a system in which other threads are executing under SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC is implementation-defined.

This policy is defined to allow strictly conforming applications to be able to indicate in a portable manner that they no longer need a realtime scheduling policy.

For threads executing under this policy, the implementation shall use only priorities within the range returned by the *sched_get_priority_max()* and *sched_get_priority_min()* functions when SCHED_OTHER is provided as the parameter.

2.8.5 Clocks and Timers

The `<time.h>` header defines the types and manifest constants used by the timing facility.

Time Value Specification Structures

Many of the timing facility functions accept or return time value specifications. A time value structure **timespec** specifies a single time value and includes at least the following members:

Member Type	Member Name	Description
time_t	<i>tv_sec</i>	Seconds.
long	<i>tv_nsec</i>	Nanoseconds.

The *tv_nsec* member is only valid if greater than or equal to zero, and less than the number of nanoseconds in a second (1 000 million). The time interval described by this structure is $(tv\_sec * 10^9 + tv\_nsec)$ nanoseconds.

A time value structure **itimerspec** specifies an initial timer value and a repetition interval for use by the per-process timer functions. This structure includes at least the following members:

Member Type	Member Name	Description
struct timespec	<i>it_interval</i>	Timer period.
struct timespec	<i>it_value</i>	Timer expiration.

If the value described by *it_value* is non-zero, it indicates the time to or time of the next timer expiration (for relative and absolute timer values, respectively). If the value described by *it_value* is zero, the timer shall be disarmed.

If the value described by *it_interval* is non-zero, it specifies an interval which shall be used in reloading the timer when it expires; that is, a periodic timer is specified. If the value described by *it_interval* is zero, the timer is disarmed after its next expiration; that is, a one-shot timer is specified.

17433 **Timer Event Notification Control Block**

17434 Per-process timers may be created that notify the process of timer expirations by queuing a
 17435 realtime extended signal. The **sigevent** structure, defined in the Base Definitions volume of
 17436 POSIX.1-2008, **<signal.h>**, is used in creating such a timer. The **sigevent** structure contains the
 17437 signal number and an application-specific data value which shall be used when notifying the
 17438 calling process of timer expiration events.

17439 **Manifest Constants**

17440 The following constants are defined in the Base Definitions volume of POSIX.1-2008, **<time.h>**:

17441 **CLOCK_REALTIME** The identifier for the system-wide realtime clock.

17442 **TIMER_ABSTIME** Flag indicating time is absolute with respect to the clock associated
 17443 with a timer.

17444 **CLOCK_MONOTONIC** The identifier for the system-wide monotonic clock, which is defined
 17445 as a clock whose value cannot be set via *clock_settime()* and which
 17446 cannot have backward clock jumps. The maximum possible clock
 17447 jump is implementation-defined.

17448 **MON** The maximum allowable resolution for **CLOCK_REALTIME** and **CLOCK_MONOTONIC** clocks
 17449 and all time services based on these clocks is represented by **{_POSIX_CLOCKRES_MIN}** and
 17450 shall be defined as 20 ms (1/50 of a second). Implementations may support smaller values of
 17451 resolution for these clocks to provide finer granularity time bases. The actual resolution
 17452 supported by an implementation for a specific clock is obtained using the *clock_getres()* function.
 17453 If the actual resolution supported for a time service based on one of these clocks differs from the
 17454 resolution supported for that clock, the implementation shall document this difference.

17455 **MON** The minimum allowable maximum value for **CLOCK_REALTIME** and **CLOCK_MONOTONIC**
 17456 clocks and all absolute time services based on them is the same as that defined by the ISO C
 17457 standard for the **time_t** type. If the maximum value supported by a time service based on one of
 17458 these clocks differs from the maximum value supported by that clock, the implementation shall
 17459 document this difference.

17460 **Execution Time Monitoring**

17461 **CPT** If **_POSIX_CPUTIME** is defined, process CPU-time clocks shall be supported in addition to the
 17462 clocks described in **Manifest Constants**.

17463 **TCT** If **_POSIX_THREAD_CPUTIME** is defined, thread CPU-time clocks shall be supported.

17464 **CPT|TCT** CPU-time clocks measure execution or CPU time, which is defined in XBD [Section 3.118](#) (on
 17465 page 52). The mechanism used to measure execution time is described in XBD [Section 4.10](#) (on
 17466 page 110).

17467 **CPT** If **_POSIX_CPUTIME** is defined, the following constant of the type **clockid_t** is defined in
 17468 **<time.h>**:

17469 **CLOCK_PROCESS_CPUTIME_ID**

17470 When this value of the type **clockid_t** is used in a *clock()* or *timer*()* function call, it is
 17471 interpreted as the identifier of the CPU-time clock associated with the process making the
 17472 function call.

17473 **TCT** If **_POSIX_THREAD_CPUTIME** is defined, the following constant of the type **clockid_t** is
 17474 defined in **<time.h>**:

CLOCK_THREAD_CPUTIME_ID

When this value of the type **clockid_t** is used in a *clock()* or *timer*()* function call, it is interpreted as the identifier of the CPU-time clock associated with the thread making the function call.

2.9 Threads

This section defines functionality to support multiple flows of control, called “threads”, within a process. For the definition of threads, see XBD [Section 3.396](#) (on page 97).

The specific functional areas covered by threads and their scope include:

- Thread management: the creation, control, and termination of multiple flows of control in the same process under the assumption of a common shared address space
- Synchronization primitives optimized for tightly coupled operation of multiple control flows in a common, shared address space

2.9.1 Thread-Safety

All functions defined by this volume of POSIX.1-2008 shall be thread-safe, except that the following functions⁷ need not be thread-safe.

<i>asctime()</i>	<i>ftw()</i>	<i>getservbyport()</i>	<i>putc_unlocked()</i>
<i>basename()</i>	<i>getc_unlocked()</i>	<i>getservent()</i>	<i>putchar_unlocked()</i>
<i>catgets()</i>	<i>getchar_unlocked()</i>	<i>getutxent()</i>	<i>putenv()</i>
<i>crypt()</i>	<i>getdate()</i>	<i>getutxid()</i>	<i>pututxline()</i>
<i>ctime()</i>	<i>getenv()</i>	<i>getutxline()</i>	<i>rand()</i>
<i>dbm_clearerr()</i>	<i>getgrent()</i>	<i>gmtime()</i>	<i>readdir()</i>
<i>dbm_close()</i>	<i>getgrgid()</i>	<i>hcreate()</i>	<i>setenv()</i>
<i>dbm_delete()</i>	<i>getgrnam()</i>	<i>hdestroy()</i>	<i>setgrent()</i>
<i>dbm_error()</i>	<i>gethostent()</i>	<i>hsearch()</i>	<i>setkey()</i>
<i>dbm_fetch()</i>	<i>getlogin()</i>	<i>inet_ntoa()</i>	<i>setpwent()</i>
<i>dbm_firstkey()</i>	<i>getnetbyaddr()</i>	<i>l64a()</i>	<i>setutxent()</i>
<i>dbm_nextkey()</i>	<i>getnetbyname()</i>	<i>lgamma()</i>	<i>strerror()</i>
<i>dbm_open()</i>	<i>getnetent()</i>	<i>lgammaf()</i>	<i>strsignal()</i>
<i>dbm_store()</i>	<i>getopt()</i>	<i>lgammal()</i>	<i>strtok()</i>
<i>dirname()</i>	<i>getprotobyname()</i>	<i>localeconv()</i>	<i>system()</i>
<i>dlderror()</i>	<i>getprotobyname()</i>	<i>localtime()</i>	<i>ttyname()</i>
<i>drand48()</i>	<i>getprotoent()</i>	<i>lrand48()</i>	<i>unsetenv()</i>
<i>encrypt()</i>	<i>getpwent()</i>	<i>mrnd48()</i>	<i>wcstombs()</i>
<i>endgrent()</i>	<i>getpwnam()</i>	<i>nftw()</i>	<i>wctomb()</i>
<i>endpwent()</i>	<i>getpwuid()</i>	<i>nl_langinfo()</i>	
<i>endutxent()</i>	<i>getservbyname()</i>	<i>ptsname()</i>	

The *ctermid()* and *tmpnam()* functions need not be thread-safe if passed a NULL argument. The *wcrtomb()* and *wcsrtombs()* functions need not be thread-safe if passed a NULL *ps* argument.

⁷ The functions in the table are not shaded to denote applicable options. Individual reference pages should be consulted.

17514 Implementations shall provide internal synchronization as necessary in order to satisfy this
17515 requirement.

17516 Since multi-threaded applications are not allowed to use the *environ* variable to access or modify
17517 any environment variable while any other thread is concurrently modifying any environment
17518 variable, any function dependent on any environment variable is not thread-safe if another
17519 thread is modifying the environment; see XSH *exec* (on page 772).

17520 2.9.2 Thread IDs

17521 Although implementations may have thread IDs that are unique in a system, applications
17522 should only assume that thread IDs are usable and unique within a single process. The effect of
17523 calling any of the functions defined in this volume of POSIX.1-2008 and passing as an argument
17524 the thread ID of a thread from another process is unspecified. The lifetime of a thread ID ends
17525 after the thread terminates if it was created with the *detachstate* attribute set to
17526 PTHREAD_CREATE_DETACHED or if *pthread_detach()* or *pthread_join()* has been called for that
17527 thread. A conforming implementation is free to reuse a thread ID after its lifetime has ended. If
17528 an application attempts to use a thread ID whose lifetime has ended, the behavior is undefined.

17529 If a thread is detached, its thread ID is invalid for use as an argument in a call to *pthread_detach()*
17530 or *pthread_join()*.

17531 2.9.3 Thread Mutexes

17532 A thread that has blocked shall not prevent any unblocked thread that is eligible to use the same
17533 processing resources from eventually making forward progress in its execution. Eligibility for
17534 processing resources is determined by the scheduling policy.

17535 A thread shall become the owner of a mutex, *m*, when one of the following occurs:

- 17536 • It returns successfully from *pthread_mutex_lock()* with *m* as the *mutex* argument.
- 17537 • It returns successfully from *pthread_mutex_trylock()* with *m* as the *mutex* argument.
- 17538 • It returns successfully from *pthread_mutex_timedlock()* with *m* as the *mutex* argument.
- 17539 • It returns (successfully or not) from *pthread_cond_wait()* with *m* as the *mutex* argument
17540 (except as explicitly indicated otherwise for certain errors).
- 17541 • It returns (successfully or not) from *pthread_cond_timedwait()* with *m* as the *mutex*
17542 argument (except as explicitly indicated otherwise for certain errors).

17543 The thread shall remain the owner of *m* until one of the following occurs:

- 17544 • It executes *pthread_mutex_unlock()* with *m* as the *mutex* argument
- 17545 • It blocks in a call to *pthread_cond_wait()* with *m* as the *mutex* argument.
- 17546 • It blocks in a call to *pthread_cond_timedwait()* with *m* as the *mutex* argument.

17547 The implementation shall behave as if at all times there is at most one owner of any mutex.

17548 A thread that becomes the owner of a mutex is said to have “acquired” the mutex and the mutex
17549 is said to have become “locked”; when a thread gives up ownership of a mutex it is said to have
17550 “released” the mutex and the mutex is said to have become “unlocked”.

17551 A problem can occur if a process terminates while one of its threads holds a mutex lock.
17552 Depending on the mutex type, it might be possible for another thread to unlock the mutex and
17553 recover the state of the mutex. However, it is difficult to perform this recovery reliably.

Robust mutexes provide a means to enable the implementation to notify other threads in the event of a process terminating while one of its threads holds a mutex lock. The next thread that acquires the mutex is notified about the termination by the return value [EOWNERDEAD] from the locking function. The notified thread can then attempt to recover the state protected by the mutex, and if successful mark the state protected by the mutex as consistent by a call to `pthread_mutex_consistent()`. If the notified thread is unable to recover the state, it can declare the state as not recoverable by a call to `pthread_mutex_unlock()` without a prior call to `pthread_mutex_consistent()`.

Whether or not the state protected by a mutex can be recovered is dependent solely on the application using robust mutexes. The robust mutex support provided in the implementation provides notification only that a mutex owner has terminated while holding a lock, or that the state of the mutex is not recoverable.

2.9.4 Thread Scheduling

The functionality described in this section shall be provided on implementations that support the Thread Execution Scheduling option (and the rest of this section is not further shaded for this option).

Thread Scheduling Attributes

In support of the scheduling function, threads have attributes which are accessed through the `pthread_attr_t` thread creation attributes object.

The *contentionscope* attribute defines the scheduling contention scope of the thread to be either `PTHREAD_SCOPE_PROCESS` or `PTHREAD_SCOPE_SYSTEM`.

The *inheritsched* attribute specifies whether a newly created thread is to inherit the scheduling attributes of the creating thread or to have its scheduling values set according to the other scheduling attributes in the `pthread_attr_t` object.

The *schedpolicy* attribute defines the scheduling policy for the thread. The *schedparam* attribute defines the scheduling parameters for the thread. The interaction of threads having different policies within a process is described as part of the definition of those policies.

If the Thread Execution Scheduling option is defined, and the *schedpolicy* attribute specifies one of the priority-based policies defined under this option, the *schedparam* attribute contains the scheduling priority of the thread. A conforming implementation ensures that the priority value in *schedparam* is in the range associated with the scheduling policy when the thread attributes object is used to create a thread, or when the scheduling attributes of a thread are dynamically modified. The meaning of the priority value in *schedparam* is the same as that of *priority*.

If `_POSIX_THREAD_SPORADIC_SERVER` is defined, the *schedparam* attribute supports four new members that are used for the sporadic server scheduling policy. These members are *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and *sched_ss_max_repl*. The meaning of these attributes is the same as in the definitions that appear under [Section 2.8.4](#) (on page 501).

When a process is created, its single thread has a scheduling policy and associated attributes equal to the policy and attributes of the process. The default scheduling contention scope value is implementation-defined. The default values of other scheduling attributes are implementation-defined.

Thread Scheduling Contention Scope

The scheduling contention scope of a thread defines the set of threads with which the thread competes for use of the processing resources. The scheduling operation selects at most one thread to execute on each processor at any point in time and the thread's scheduling attributes (for example, *priority*), whether under process scheduling contention scope or system scheduling contention scope, are the parameters used to determine the scheduling decision.

The scheduling contention scope, in the context of scheduling a mixed scope environment, affects threads as follows:

- A thread created with PTHREAD_SCOPE_SYSTEM scheduling contention scope contends for resources with all other threads in the same scheduling allocation domain relative to their system scheduling attributes. The system scheduling attributes of a thread created with PTHREAD_SCOPE_SYSTEM scheduling contention scope are the scheduling attributes with which the thread was created. The system scheduling attributes of a thread created with PTHREAD_SCOPE_PROCESS scheduling contention scope are the implementation-defined mapping into system attribute space of the scheduling attributes with which the thread was created.
- Threads created with PTHREAD_SCOPE_PROCESS scheduling contention scope contend directly with other threads within their process that were created with PTHREAD_SCOPE_PROCESS scheduling contention scope. The contention is resolved based on the threads' scheduling attributes and policies. It is unspecified how such threads are scheduled relative to threads in other processes or threads with PTHREAD_SCOPE_SYSTEM scheduling contention scope.
- Conforming implementations shall support the PTHREAD_SCOPE_PROCESS scheduling contention scope, the PTHREAD_SCOPE_SYSTEM scheduling contention scope, or both.

Scheduling Allocation Domain

Implementations shall support scheduling allocation domains containing one or more processors. It should be noted that the presence of multiple processors does not automatically indicate a scheduling allocation domain size greater than one. Conforming implementations on multi-processors may map all or any subset of the CPUs to one or multiple scheduling allocation domains, and could define these scheduling allocation domains on a per-thread, per-process, or per-system basis, depending on the types of applications intended to be supported by the implementation. The scheduling allocation domain is independent of scheduling contention scope, as the scheduling contention scope merely defines the set of threads with which a thread contends for processor resources, while scheduling allocation domain defines the set of processors for which it contends. The semantics of how this contention is resolved among threads for processors is determined by the scheduling policies of the threads.

The choice of scheduling allocation domain size and the level of application control over scheduling allocation domains is implementation-defined. Conforming implementations may change the size of scheduling allocation domains and the binding of threads to scheduling allocation domains at any time.

For application threads with scheduling allocation domains of size equal to one, the scheduling rules defined for SCHED_FIFO and SCHED_RR shall be used; see [Scheduling Policies](#) (on page 501). All threads with system scheduling contention scope, regardless of the processes in which they reside, compete for the processor according to their priorities. Threads with process scheduling contention scope compete only with other threads with process scheduling contention scope within their process.

For application threads with scheduling allocation domains of size greater than one, the rules

defined for SCHED_FIFO, SCHED_RR, and SCHED_SPORADIC shall be used in an implementation-defined manner. Each thread with system scheduling contention scope competes for the processors in its scheduling allocation domain in an implementation-defined manner according to its priority. Threads with process scheduling contention scope are scheduled relative to other threads within the same scheduling contention scope in the process.

If _POSIX_THREAD_SPORADIC_SERVER is defined, the rules defined for SCHED_SPORADIC in *Scheduling Policies* (on page 501) shall be used in an implementation-defined manner for application threads whose scheduling allocation domain size is greater than one.

Scheduling Documentation

If _POSIX_PRIORITY_SCHEDULING is defined, then any scheduling policies beyond SCHED_OTHER, SCHED_FIFO, SCHED_RR, and SCHED_SPORADIC, as well as the effects of the scheduling policies indicated by these other values, and the attributes required in order to support such a policy, are implementation-defined. Furthermore, the implementation shall document the effect of all processor scheduling allocation domain values supported for these policies.

2.9.5 Thread Cancellation

The thread cancellation mechanism allows a thread to terminate the execution of any other thread in the process in a controlled manner. The target thread (that is, the one that is being canceled) is allowed to hold cancellation requests pending in a number of ways and to perform application-specific cleanup processing when the notice of cancellation is acted upon.

Cancellation is controlled by the cancellation control functions. Each thread maintains its own cancelability state. Cancellation may only occur at cancellation points or when the thread is asynchronously cancelable.

The thread cancellation mechanism described in this section depends upon programs having set *deferred* cancelability state, which is specified as the default. Applications shall also carefully follow static lexical scoping rules in their execution behavior. For example, use of *setjmp()*, *return*, *goto*, and so on, to leave user-defined cancellation scopes without doing the necessary scope pop operation results in undefined behavior.

Use of asynchronous cancelability while holding resources which potentially need to be released may result in resource loss. Similarly, cancellation scopes may only be safely manipulated (pushed and popped) when the thread is in the *deferred* or *disabled* cancelability states.

2.9.5.1 Cancelability States

The cancelability state of a thread determines the action taken upon receipt of a cancellation request. The thread may control cancellation in a number of ways.

Each thread maintains its own cancelability state, which may be encoded in two bits:

1. Cancelability-Enable: When cancelability is PTHREAD_CANCEL_DISABLE (as defined in the Base Definitions volume of POSIX.1-2008, *<pthread.h>*), cancellation requests against the target thread are held pending. By default, cancelability is set to PTHREAD_CANCEL_ENABLE (as defined in *<pthread.h>*).
2. Cancelability Type: When cancelability is enabled and the cancelability type is PTHREAD_CANCEL_ASYNCHRONOUS (as defined in *<pthread.h>*), new or pending cancellation requests may be acted upon at any time. When cancelability is enabled and the cancelability type is PTHREAD_CANCEL_DEFERRED (as defined in *<pthread.h>*),

17686 cancellation requests are held pending until a cancellation point (see below) is reached. If
 17687 cancelability is disabled, the setting of the cancelability type has no immediate effect as all
 17688 cancellation requests are held pending; however, once cancelability is enabled again the
 17689 new type is in effect. The cancelability type is PTHREAD_CANCEL_DEFERRED in all
 17690 newly created threads including the thread in which *main()* was first invoked.

17691 2.9.5.2 Cancellation Points

17692 Cancellation points shall occur when a thread is executing the following functions:

17693	<i>accept()</i>	<i>nanosleep()</i>	<i>select()</i>
17694	<i>aio_suspend()</i>	<i>open()</i>	<i>sem_timedwait()</i>
17695	<i>clock_nanosleep()</i>	<i>openat()</i>	<i>sem_wait()</i>
17696	<i>close()</i>	<i>pause()</i>	<i>send()</i>
17697	<i>connect()</i>	<i>poll()</i>	<i>sendmsg()</i>
17698	<i>creat()</i>	<i>pread()</i>	<i>sendto()</i>
17699	<i>fcntl()</i> [†]	<i>pselect()</i>	<i>sigsuspend()</i>
17700	<i>fdatasync()</i>	<i>pthread_cond_timedwait()</i>	<i>sigtimedwait()</i>
17701	<i>fsync()</i>	<i>pthread_cond_wait()</i>	<i>sigwait()</i>
17702	<i>getmsg()</i>	<i>pthread_join()</i>	<i>sigwaitinfo()</i>
17703	<i>getpmsg()</i>	<i>pthread_testcancel()</i>	<i>sleep()</i>
17704	<i>lockf()</i> ^{††}	<i>putmsg()</i>	<i>system()</i>
17705	<i>mq_receive()</i>	<i>putpmsg()</i>	<i>tcdrain()</i>
17706	<i>mq_send()</i>	<i>pwrite()</i>	<i>wait()</i>
17707	<i>mq_timedreceive()</i>	<i>read()</i>	<i>waitid()</i>
17708	<i>mq_timedsend()</i>	<i>readv()</i>	<i>waitpid()</i>
17709	<i>msgrcv()</i>	<i>recv()</i>	<i>write()</i>
17710	<i>msgsnd()</i>	<i>recvfrom()</i>	<i>writew()</i>
17711	<i>msync()</i>	<i>recvmsg()</i>	

17712 [†] When the *cmd* argument is F_SETLK.

17713 ^{††} When the *function* argument is F_LOCK.

General Information

Threads

17714 A cancellation point may also occur when a thread is executing the following functions:

17715	<i>access()</i>	<i>fprintf()</i>	<i>getprotobyname()</i>
17716	<i>asctime()</i>	<i>fputc()</i>	<i>getprotoent()</i>
17717	<i>asctime_r()</i>	<i>fputs()</i>	<i>getpwent()</i>
17718	<i>catclose()</i>	<i>fputwc()</i>	<i>getpwnam()</i>
17719	<i>catgets()</i>	<i>fputws()</i>	<i>getpwnam_r()</i>
17720	<i>catopen()</i>	<i>fread()</i>	<i>getpwuid()</i>
17721	<i>chmod()</i>	<i>freopen()</i>	<i>getpwuid_r()</i>
17722	<i>chown()</i>	<i>fscanf()</i>	<i>gets()</i>
17723	<i>closedir()</i>	<i>fseek()</i>	<i>getservbyname()</i>
17724	<i>closelog()</i>	<i>fseeko()</i>	<i>getservbyport()</i>
17725	<i>ctermid()</i>	<i>fsetpos()</i>	<i>getservent()</i>
17726	<i>ctime()</i>	<i>fstat()</i>	<i>getutxent()</i>
17727	<i>ctime_r()</i>	<i>fstatat()</i>	<i>getutxid()</i>
17728	<i>dbm_close()</i>	<i>ftell()</i>	<i>getutxline()</i>
17729	<i>dbm_delete()</i>	<i>ftello()</i>	<i>getwc()</i>
17730	<i>dbm_fetch()</i>	<i>ftw()</i>	<i>getwchar()</i>
17731	<i>dbm_nextkey()</i>	<i>futimens()</i>	<i>glob()</i>
17732	<i>dbm_open()</i>	<i>fwprintf()</i>	<i>iconv_close()</i>
17733	<i>dbm_store()</i>	<i>fwrite()</i>	<i>iconv_open()</i>
17734	<i>dlclose()</i>	<i>fwscanf()</i>	<i>ioctl()</i>
17735	<i>dlopen()</i>	<i>getaddrinfo()</i>	<i>link()</i>
17736	<i>dprintf()</i>	<i>getc()</i>	<i>linkat()</i>
17737	<i>endgrent()</i>	<i>getc_unlocked()</i>	<i>lio_listio()</i>
17738	<i>endhostent()</i>	<i>getchar()</i>	<i>localtime()</i>
17739	<i>endnetent()</i>	<i>getchar_unlocked()</i>	<i>localtime_r()</i>
17740	<i>endprotoent()</i>	<i>getcwd()</i>	<i>lockf()</i>
17741	<i>endpwent()</i>	<i>getdate()</i>	<i>lseek()</i>
17742	<i>endservent()</i>	<i>getdelim()</i>	<i>lstat()</i>
17743	<i>endutxent()</i>	<i>getgrent()</i>	<i>mkdir()</i>
17744	<i>faccessat()</i>	<i>getgrgid()</i>	<i>mkdirat()</i>
17745	<i>fchmod()</i>	<i>getgrgid_r()</i>	<i>mkdtemp()</i>
17746	<i>fchmodat()</i>	<i>getgrnam()</i>	<i>mkfifo()</i>
17747	<i>fchown()</i>	<i>getgrnam_r()</i>	<i>mkfifoat()</i>
17748	<i>fchownat()</i>	<i>gethostent()</i>	<i>mknod()</i>
17749	<i>fclose()</i>	<i>gethostid()</i>	<i>mknodat()</i>
17750	<i>fcntl()</i> †	<i>gethostname()</i>	<i>mkstemp()</i>
17751	<i>fflush()</i>	<i>getline()</i>	<i>mktime()</i>
17752	<i>fgetc()</i>	<i>getlogin()</i>	<i>nftw()</i>
17753	<i>fgetpos()</i>	<i>getlogin_r()</i>	<i>opendir()</i>
17754	<i>fgets()</i>	<i>getnameinfo()</i>	<i>openlog()</i>
17755	<i>fgetwc()</i>	<i>getnetbyaddr()</i>	<i>pathconf()</i>
17756	<i>fgetws()</i>	<i>getnetbyname()</i>	<i>pclose()</i>
17757	<i>fntmsg()</i>	<i>getnetent()</i>	<i>perror()</i>
17758	<i>fopen()</i>	<i>getopt()</i> ††	<i>popen()</i>
17759	<i>fpathconf()</i>	<i>getprotobyname()</i>	<i>posix_fadvise()</i>

17760	<code>posix_fallocate()</code>	<code>putc()</code>	<code>strerror()</code>
17761	<code>posix_madvise()</code>	<code>putc_unlocked()</code>	<code>strerror_r()</code>
17762	<code>posix_openpt()</code>	<code>putchar()</code>	<code>strftime()</code>
17763	<code>posix_spawn()</code>	<code>putchar_unlocked()</code>	<code>symlink()</code>
17764	<code>posix_spawnnp()</code>	<code>puts()</code>	<code>symlinkat()</code>
17765	<code>posix_trace_clear()</code>	<code>pututxline()</code>	<code>sync()</code>
17766	<code>posix_trace_close()</code>	<code>putwc()</code>	<code>syslog()</code>
17767	<code>posix_trace_create()</code>	<code>putwchar()</code>	<code>tmpfile()</code>
17768	<code>posix_trace_create_withlog()</code>	<code>readdir()</code>	<code>tmpnam()</code>
17769	<code>posix_trace_eventtypelist_getnext_id()</code>	<code>readdir_r()</code>	<code>ttyname()</code>
17770	<code>posix_trace_eventtypelist_rewind()</code>	<code>readlink()</code>	<code>ttyname_r()</code>
17771	<code>posix_trace_flush()</code>	<code>readlinkat()</code>	<code>tzset()</code>
17772	<code>posix_trace_get_attr()</code>	<code>remove()</code>	<code>ungetc()</code>
17773	<code>posix_trace_get_filter()</code>	<code>rename()</code>	<code>ungetwc()</code>
17774	<code>posix_trace_get_status()</code>	<code>renameat()</code>	<code>unlink()</code>
17775	<code>posix_trace_getnext_event()</code>	<code>rewind()</code>	<code>unlinkat()</code>
17776	<code>posix_trace_open()</code>	<code>rewinddir()</code>	<code>utime()</code>
17777	<code>posix_trace_rewind()</code>	<code>scandir()</code>	<code>utimensat()</code>
17778	<code>posix_trace_set_filter()</code>	<code>scanf()</code>	<code>utimes()</code>
17779	<code>posix_trace_shutdown()</code>	<code>seekdir()</code>	<code>vdprintf()</code>
17780	<code>posix_trace_timedgetnext_event()</code>	<code>semop()</code>	<code>vfprintf()</code>
17781	<code>posix_typed_mem_open()</code>	<code>setgrent()</code>	<code>vfwprintf()</code>
17782	<code>printf()</code>	<code>sethostent()</code>	<code>vprintf()</code>
17783	<code>psiginfo()</code>	<code>setnetent()</code>	<code>vwprintf()</code>
17784	<code>psignal()</code>	<code>setprotoent()</code>	<code>wcsftime()</code>
17785	<code>pthread_rwlock_rdlock()</code>	<code>setpwent()</code>	<code>wordexp()</code>
17786	<code>pthread_rwlock_timedrdlock()</code>	<code>setserverent()</code>	<code>wprintf()</code>
17787	<code>pthread_rwlock_timedwrlock()</code>	<code>setutxent()</code>	<code>wscanf()</code>
17788	<code>pthread_rwlock_wrlock()</code>	<code>sigpause()</code>	
17789		<code>stat()</code>	

An implementation shall not introduce cancellation points into any other functions specified in this volume of POSIX.1-2008.

The side-effects of acting upon a cancellation request while suspended during a call of a function are the same as the side-effects that may be seen in a single-threaded program when a call to a function is interrupted by a signal and the given function returns [EINTR]. Any such side-effects occur before any cancellation cleanup handlers are called.

Whenever a thread has cancelability enabled and a cancellation request has been made with that thread as the target, and the thread then calls any function that is a cancellation point (such as `pthread_testcancel()` or `read()`), the cancellation request shall be acted upon before the function returns. If a thread has cancelability enabled and a cancellation request is made with the thread as a target while the thread is suspended at a cancellation point, the thread shall be awakened and the cancellation request shall be acted upon. It is unspecified whether the cancellation request is acted upon or whether the cancellation request remains pending and the thread resumes normal execution if:

- The thread is suspended at a cancellation point and the event for which it is waiting occurs

† For any value of the *cmd* argument.

†† If *opterr* is non-zero.

- A specified timeout expired
- before the cancellation request is acted upon.

2.9.5.3 Thread Cancellation Cleanup Handlers

Each thread maintains a list of cancellation cleanup handlers. The programmer uses the *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions to place routines on and remove routines from this list.

When a cancellation request is acted upon, or when a thread calls *pthread_exit()*, the thread first disables cancellation by setting its cancelability state to *PTHREAD_CANCEL_DISABLE* and its cancelability type to *PTHREAD_CANCEL_DEFERRED*. The cancelability state shall remain set to *PTHREAD_CANCEL_DISABLE* until the thread has terminated. The behavior is undefined if a cancellation cleanup handler or thread-specific data destructor routine changes the cancelability state to *PTHREAD_CANCEL_ENABLE*.

The routines in the thread's list of cancellation cleanup handlers are invoked one by one in LIFO sequence; that is, the last routine pushed onto the list (Last In) is the first to be invoked (First Out). When the cancellation cleanup handler for a scope is invoked, the storage for that scope remains valid. If the last cancellation cleanup handler returns, thread-specific data destructors (if any) associated with thread-specific data keys for which the thread has non-NULL values will be run, in unspecified order, as described for *pthread_key_create()*.

After all cancellation cleanup handlers and thread-specific data destructors have returned, thread execution is terminated. If the thread has terminated because of a call to *pthread_exit()*, the *value_ptr* argument is made available to any threads joining with the target. If the thread has terminated by acting on a cancellation request, a status of *PTHREAD_CANCELED* is made available to any threads joining with the target. The symbolic constant *PTHREAD_CANCELED* expands to a constant expression of type (**void ***) whose value matches no pointer to an object in memory nor the value *NULL*.

A side-effect of acting upon a cancellation request while in a condition variable wait is that the mutex is re-acquired before calling the first cancellation cleanup handler. In addition, the thread is no longer considered to be waiting for the condition and the thread shall not have consumed any pending condition signals on the condition.

A cancellation cleanup handler cannot exit via *longjmp()* or *siglongjmp()*.

2.9.5.4 Async-Cancel Safety

The *pthread_cancel()*, *pthread_setcancelstate()*, and *pthread_setcanceltype()* functions are defined to be async-cancel safe.

No other functions in this volume of POSIX.1-2008 are required to be async-cancel-safe.

2.9.6 Thread Read-Write Locks

Multiple readers, single writer (read-write) locks allow many threads to have simultaneous read-only access to data while allowing only one thread to have exclusive write access at any given time. They are typically used to protect data that is read more frequently than it is changed.

One or more readers acquire read access to the resource by performing a read lock operation on the associated read-write lock. A writer acquires exclusive write access by performing a write lock operation. Basically, all readers exclude any writers and a writer excludes all readers and

any other writers.

A thread that has blocked on a read-write lock (for example, has not yet returned from a *pthread_rwlock_rdlock()* or *pthread_rwlock_wrlock()* call) shall not prevent any unblocked thread that is eligible to use the same processing resources from eventually making forward progress in its execution. Eligibility for processing resources shall be determined by the scheduling policy.

Read-write locks can be used to synchronize threads in the current process and other processes if they are allocated in memory that is writable and shared among the cooperating processes and have been initialized for this behavior.

2.9.7 Thread Interactions with Regular File Operations

All of the following functions shall be atomic with respect to each other in the effects specified in POSIX.1-2008 when they operate on regular files or symbolic links:

<i>chmod()</i>	<i>fchownat()</i>	<i>lseek()</i>	<i>readv()</i>	<i>unlink()</i>
<i>chown()</i>	<i>fcntl()</i>	<i>lstat()</i>	<i>pwrite()</i>	<i>unlinkat()</i>
<i>close()</i>	<i>fstat()</i>	<i>open()</i>	<i>rename()</i>	<i>utime()</i>
<i>creat()</i>	<i>fstatat()</i>	<i>openat()</i>	<i>renameat()</i>	<i>utimensat()</i>
<i>dup2()</i>	<i>ftruncate()</i>	<i>pread()</i>	<i>stat()</i>	<i>utimes()</i>
<i>fchmod()</i>	<i>lchown()</i>	<i>read()</i>	<i>symlink()</i>	<i>write()</i>
<i>fchmodat()</i>	<i>link()</i>	<i>readlink()</i>	<i>symlinkat()</i>	<i>writev()</i>
<i>fchown()</i>	<i>linkat()</i>	<i>readlinkat()</i>	<i>truncate()</i>	

If two threads each call one of these functions, each call shall either see all of the specified effects of the other call, or none of them.

2.9.8 Use of Application-Managed Thread Stacks

An “application-managed thread stack” is a region of memory allocated by the application—for example, memory returned by the *malloc()* or *mmap()* functions—and designated as a stack through the act of passing the address and size of the stack, respectively, as the *stackaddr* and *stacksize* arguments to *pthread_attr_setstack()*. Application-managed stacks allow the application to precisely control the placement and size of a stack.

The application grants to the implementation permanent ownership of and control over the application-managed stack when the attributes object in which the *stack* or *stackaddr* attribute has been set is used, either by presenting that attribute’s object as the *attr* argument in a call to *pthread_create()* that completes successfully, or by storing a pointer to the attributes object in the *sigev_notify_attributes* member of a **struct sigevent** and passing that **struct sigevent** to a function accepting such argument that completes successfully. The application may thereafter utilize the memory within the stack only within the normal context of stack usage within or properly synchronized with a thread that has been scheduled by the implementation with stack pointer value(s) that are within the range of that stack. In particular, the region of memory cannot be freed, nor can it be later specified as the stack for another thread.

When specifying an attributes object with an application-managed stack through the *sigev_notify_attributes* member of a **struct sigevent**, the results are undefined if the requested signal is generated multiple times (as for a repeating timer).

Until an attributes object in which the *stack* or *stackaddr* attribute has been set is used, the application retains ownership of and control over the memory allocated to the stack. It may free or reuse the memory as long as it either deletes the attributes object, or before using the

attributes object replaces the stack by making an additional call to `pthread_attr_setstack()`, that was used originally to designate the stack. There is no mechanism to retract the reference to an application-managed stack by an existing attributes object.

Once an attributes object with an application-managed stack has been used, that attributes object cannot be used again by a subsequent call to `pthread_create()` or any function accepting a **struct sigevent** with `sigev_notify_attributes` containing a pointer to the attributes object, without designating an unused application-managed stack by making an additional call to `pthread_attr_setstack()`.

2.10 Sockets

A socket is an endpoint for communication using the facilities described in this section. A socket is created with a specific socket type, described in [Section 2.10.6](#) (on page 518), and is associated with a specific protocol, detailed in [Section 2.10.3](#). A socket is accessed via a file descriptor obtained when the socket is created.

2.10.1 Address Families

All network protocols are associated with a specific address family. An address family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. An address family is normally comprised of a number of protocols, one per socket type. Each protocol is characterized by an abstract socket type. It is not required that an address family support all socket types. An address family may contain multiple protocols supporting the same socket abstraction.

[Section 2.10.17](#) (on page 525), [Section 2.10.19](#) (on page 526), and [Section 2.10.20](#) (on page 526), respectively, describe the use of sockets for local UNIX connections, for Internet protocols based on IPv4, and for Internet protocols based on IPv6.

2.10.2 Addressing

An address family defines the format of a socket address. All network addresses are described using a general structure, called a **sockaddr**, as defined in the Base Definitions volume of POSIX.1-2008, `<sys/socket.h>`. However, each address family imposes finer and more specific structure, generally defining a structure with fields specific to the address family. The field `sa_family` in the **sockaddr** structure contains the address family identifier, specifying the format of the `sa_data` area. The size of the `sa_data` area is unspecified.

2.10.3 Protocols

A protocol supports one of the socket abstractions detailed in [Section 2.10.6](#) (on page 518). Selecting a protocol involves specifying the address family, socket type, and protocol number to the `socket()` function. Certain semantics of the basic socket abstractions are protocol-specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide non-standard facilities or extensions to a mechanism.

17929 **2.10.4 Routing**

17930 Sockets provides packet routing facilities. A routing information database is maintained, which
 17931 is used in selecting the appropriate network interface when transmitting packets.

17932 **2.10.5 Interfaces**

17933 Each network interface in a system corresponds to a path through which messages can be sent
 17934 and received. A network interface usually has a hardware device associated with it, though
 17935 certain interfaces such as the loopback interface, do not.

17936 **2.10.6 Socket Types**

17937 A socket is created with a specific type, which defines the communication semantics and which
 17938 allows the selection of an appropriate communication protocol. Four types are defined:
 17939 RS SOCK_DGRAM, SOCK_RAW, SOCK_SEQPACKET, and SOCK_STREAM. Implementations
 17940 may specify additional socket types.

17941 The SOCK_STREAM socket type provides reliable, sequenced, full-duplex octet streams
 17942 between the socket and a peer to which the socket is connected. A socket of type
 17943 SOCK_STREAM must be in a connected state before any data may be sent or received. Record
 17944 boundaries are not maintained; data sent on a stream socket using output operations of one size
 17945 may be received using input operations of smaller or larger sizes without loss of data. Data may
 17946 be buffered; successful return from an output function does not imply that the data has been
 17947 delivered to the peer or even transmitted from the local system. If data cannot be successfully
 17948 transmitted within a given time then the connection is considered broken, and subsequent
 17949 operations shall fail. A SIGPIPE signal is raised if a thread attempts to send data on a broken
 17950 stream (one that is no longer connected), except that the signal is suppressed if the
 17951 MSG_NOSIGNAL flag is used in calls to *send()*, *sendto()*, and *sendmsg()*. Support for an out-of-
 17952 band data transmission facility is protocol-specific.

17953 The SOCK_SEQPACKET socket type is similar to the SOCK_STREAM type, and is also
 17954 connection-oriented. The only difference between these types is that record boundaries are
 17955 maintained using the SOCK_SEQPACKET type. A record can be sent using one or more output
 17956 operations and received using one or more input operations, but a single operation never
 17957 transfers parts of more than one record. Record boundaries are visible to the receiver via the
 17958 MSG_EOR flag in the received message flags returned by the *recvmsg()* function. It is protocol-
 17959 specific whether a maximum record size is imposed.

17960 The SOCK_DGRAM socket type supports connectionless data transfer which is not necessarily
 17961 acknowledged or reliable. Datagrams may be sent to the address specified (possibly multicast or
 17962 broadcast) in each output operation, and incoming datagrams may be received from multiple
 17963 sources. The source address of each datagram is available when receiving the datagram. An
 17964 application may also pre-specify a peer address, in which case calls to output functions that do
 17965 not specify a peer address shall send to the pre-specified peer. If a peer has been specified, only
 17966 datagrams from that peer shall be received. A datagram must be sent in a single output
 17967 operation, and must be received in a single input operation. The maximum size of a datagram is
 17968 protocol-specific; with some protocols, the limit is implementation-defined. Output datagrams
 17969 may be buffered within the system; thus, a successful return from an output function does not
 17970 guarantee that a datagram is actually sent or received. However, implementations should
 17971 attempt to detect any errors possible before the return of an output function, reporting any error
 17972 by an unsuccessful return value.

17973 RS The SOCK_RAW socket type is similar to the SOCK_DGRAM type. It differs in that it is
 17974 normally used with communication providers that underlie those used for the other socket
 17975 types. For this reason, the creation of a socket with type SOCK_RAW shall require appropriate
 17976 privileges. The format of datagrams sent and received with this socket type generally include
 17977 specific protocol headers, and the formats are protocol-specific and implementation-defined.

17978 2.10.7 Socket I/O Mode

17979 The I/O mode of a socket is described by the O_NONBLOCK file status flag which pertains to
 17980 the open file description for the socket. This flag is initially off when a socket is created, but may
 17981 be set and cleared by the use of the F_SETFL command of the *fcntl()* function.

17982 When the O_NONBLOCK flag is set, certain functions that would normally block until they are
 17983 complete shall return immediately.

17984 The *bind()* function initiates an address assignment and shall return without blocking when
 17985 O_NONBLOCK is set; if the socket address cannot be assigned immediately, *bind()* shall return
 17986 the [EINPROGRESS] error to indicate that the assignment was initiated successfully, but that it
 17987 has not yet completed.

17988 The *connect()* function initiates a connection and shall return without blocking when
 17989 O_NONBLOCK is set; it shall return the error [EINPROGRESS] to indicate that the connection
 17990 was initiated successfully, but that it has not yet completed.

17991 Data transfer operations (the *read()*, *write()*, *send()*, and *recv()* functions) shall complete
 17992 immediately, transfer only as much as is available, and then return without blocking, or return
 17993 an error indicating that no transfer could be made without blocking.

17994 2.10.8 Socket Owner

17995 The owner of a socket is unset when a socket is created. The owner may be set to a process ID or
 17996 process group ID using the F_SETOWN command of the *fcntl()* function.

17997 2.10.9 Socket Queue Limits

17998 The transmit and receive queue sizes for a socket are set when the socket is created. The default
 17999 sizes used are both protocol-specific and implementation-defined. The sizes may be changed
 18000 using the *setsockopt()* function.

18001 2.10.10 Pending Error

18002 Errors may occur asynchronously, and be reported to the socket in response to input from the
 18003 network protocol. The socket stores the pending error to be reported to a user of the socket at the
 18004 next opportunity. The error is returned in response to a subsequent *send()*, *recv()*, or *getsockopt()*
 18005 operation on the socket, and the pending error is then cleared.

2.10.11 Socket Receive Queue

A socket has a receive queue that buffers data when it is received by the system until it is removed by a receive call. Depending on the type of the socket and the communication provider, the receive queue may also contain ancillary data such as the addressing and other protocol data associated with the normal data in the queue, and may contain out-of-band or expedited data. The limit on the queue size includes any normal, out-of-band data, datagram source addresses, and ancillary data in the queue. The description in this section applies to all sockets, even though some elements cannot be present in some instances.

The contents of a receive buffer are logically structured as a series of data segments with associated ancillary data and other information. A data segment may contain normal data or out-of-band data, but never both. A data segment may complete a record if the protocol supports records (always true for types SOCK_SEQPACKET and SOCK_DGRAM). A record may be stored as more than one segment; the complete record might never be present in the receive buffer at one time, as a portion might already have been returned to the application, and another portion might not yet have been received from the communications provider. A data segment may contain ancillary protocol data, which is logically associated with the segment. Ancillary data is received as if it were queued along with the first normal data octet in the segment (if any). A segment may contain ancillary data only, with no normal or out-of-band data. For the purposes of this section, a datagram is considered to be a data segment that terminates a record, and that includes a source address as a special type of ancillary data. Data segments are placed into the queue as data is delivered to the socket by the protocol. Normal data segments are placed at the end of the queue as they are delivered. If a new segment contains the same type of data as the preceding segment and includes no ancillary data, and if the preceding segment does not terminate a record, the segments are logically merged into a single segment.

The receive queue is logically terminated if an end-of-file indication has been received or a connection has been terminated. A segment shall be considered to be terminated if another segment follows it in the queue, if the segment completes a record, or if an end-of-file or other connection termination has been reported. The last segment in the receive queue shall also be considered to be terminated while the socket has a pending error to be reported.

A receive operation shall never return data or ancillary data from more than one segment.

2.10.12 Socket Out-of-Band Data State

The handling of received out-of-band data is protocol-specific. Out-of-band data may be placed in the socket receive queue, either at the end of the queue or before all normal data in the queue. In this case, out-of-band data is returned to an application program by a normal receive call. Out-of-band data may also be queued separately rather than being placed in the socket receive queue, in which case it shall be returned only in response to a receive call that requests out-of-band data. It is protocol-specific whether an out-of-band data mark is placed in the receive queue to demarcate data preceding the out-of-band data and following the out-of-band data. An out-of-band data mark is logically an empty data segment that cannot be merged with other segments in the queue. An out-of-band data mark is never returned in response to an input operation. The *socketatmark()* function can be used to test whether an out-of-band data mark is the first element in the queue. If an out-of-band data mark is the first element in the queue when an input function is called without the MSG_PEEK option, the mark is removed from the queue and the following data (if any) is processed as if the mark had not been present.

2.10.13 Connection Indication Queue

Sockets that are used to accept incoming connections maintain a queue of outstanding connection indications. This queue is a list of connections that are awaiting acceptance by the application; see *listen()*.

2.10.14 Signals

One category of event at the socket interface is the generation of signals. These signals report protocol events or process errors relating to the state of the socket. The generation or delivery of a signal does not change the state of the socket, although the generation of the signal may have been caused by a state change.

The SIGPIPE signal shall be sent to a thread that attempts to send data on a socket that is no longer able to send (one that is no longer connected), except that the signal is suppressed if the MSG_NOSIGNAL flag is used in calls to *send()*, *sendto()*, and *sendmsg()*. Regardless of whether the generation of the signal is suppressed, the send operation shall fail with the [EPIPE] error.

If a socket has an owner, the SIGURG signal is sent to the owner of the socket when it is notified of expedited or out-of-band data. The socket state at this time is protocol-dependent, and the status of the socket is specified in [Section 2.10.17](#) (on page 525), [Section 2.10.19](#) (on page 526), and [Section 2.10.20](#) (on page 526). Depending on the protocol, the expedited data may or may not have arrived at the time of signal generation.

2.10.15 Asynchronous Errors

If any of the following conditions occur asynchronously for a socket, the corresponding value listed below shall become the pending error for the socket:

[ECONNABORTED]

The connection was aborted locally.

[ECONNREFUSED]

For a connection-mode socket attempting a non-blocking connection, the attempt to connect was forcefully rejected. For a connectionless-mode socket, an attempt to deliver a datagram was forcefully rejected.

[ECONNRESET]

The peer has aborted the connection.

[EHOSTDOWN]

The destination host has been determined to be down or disconnected.

[EHOSTUNREACH]

The destination host is not reachable.

[EMSGSIZE]

For a connectionless-mode socket, the size of a previously sent datagram prevented delivery.

[ENETDOWN]

The local network connection is not operational.

[ENETRESET]

The connection was aborted by the network.

18091 [ENETUNREACH]
18092 The destination network is not reachable.

18093 2.10.16 Use of Options

18094 There are a number of socket options which either specialize the behavior of a socket or provide
18095 useful information. These options may be set at different protocol levels and are always present
18096 at the uppermost “socket” level.

18097 Socket options are manipulated by two functions, *getsockopt()* and *setsockopt()*. These functions
18098 allow an application program to customize the behavior and characteristics of a socket to
18099 provide the desired effect.

18100 All of the options have default values. The type and meaning of these values is defined by the
18101 protocol level to which they apply. Instead of using the default values, an application program
18102 may choose to customize one or more of the options. However, in the bulk of cases, the default
18103 values are sufficient for the application.

18104 Some of the options are used to enable or disable certain behavior within the protocol modules
18105 (for example, turn on debugging) while others may be used to set protocol-specific information
18106 (for example, IP time-to-live on all the application’s outgoing packets). As each of the options is
18107 introduced, its effect on the underlying protocol modules is described.

18108 Table 2-1 shows the value for the socket level.

18109 Table 2-1 Value of Level for Socket Options

18110	Name	Description
18111	SOL_SOCKET	Options are intended for the sockets level.

18112 Table 2-2 (on page 523) lists those options present at the socket level; that is, when the *level*
18113 parameter of the *getsockopt()* or *setsockopt()* function is SOL_SOCKET, the types of the option
18114 value parameters associated with each option, and a brief synopsis of the meaning of the option
18115 value parameter. Unless otherwise noted, each may be examined with *getsockopt()* and set with
18116 *setsockopt()* on all types of socket. Options at other protocol levels vary in format and name.

Table 2-2 Socket-Level Options

Option	Parameter Type	Parameter Meaning
SO_ACCEPTCONN	int	Non-zero indicates that socket listening is enabled (<i>getsockopt()</i> only).
SO_BROADCAST	int	Non-zero requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).
SO_DEBUG	int	Non-zero requests debugging in underlying protocol modules.
SO_DONTROUTE	int	Non-zero requests bypass of normal routing; route based on destination address only.
SO_ERROR	int	Requests and clears pending error information on the socket (<i>getsockopt()</i> only).
SO_KEEPALIVE	int	Non-zero requests periodic transmission of keepalive messages (protocol-specific).
SO_LINGER	struct linger	Specify actions to be taken for queued, unsent data on <i>close()</i> : linger on/off and linger time in seconds.
SO_OOINLINE	int	Non-zero requests that out-of-band data be placed into normal data input queue as received.
SO_RCVBUF	int	Size of receive buffer (in bytes).
SO_RCVLOWAT	int	Minimum amount of data to return to application for input operations (in bytes).
SO_RCVTIMEO	struct timeval	Timeout value for a socket receive operation.
SO_REUSEADDR	int	Non-zero requests reuse of local addresses in <i>bind()</i> (protocol-specific).
SO_SNDBUF	int	Size of send buffer (in bytes).
SO_SNDLOWAT	int	Minimum amount of data to send for output operations (in bytes).
SO_SNDTIMEO	struct timeval	Timeout value for a socket send operation.
SO_TYPE	int	Identify socket type (<i>getsockopt()</i> only).

The SO_ACCEPTCONN option is used only on *getsockopt()*. When this option is specified, *getsockopt()* shall report whether socket listening is enabled for the socket. A value of zero shall indicate that socket listening is disabled; non-zero that it is enabled. SO_ACCEPTCONN has no default value.

The SO_BROADCAST option requests permission to send broadcast datagrams on the socket. Support for SO_BROADCAST is protocol-specific. The default for SO_BROADCAST is that the ability to send broadcast datagrams on a socket is disabled.

The SO_DEBUG option enables debugging in the underlying protocol modules. This can be useful for tracing the behavior of the underlying protocol modules during normal system operation. The semantics of the debug reports are implementation-defined. The default value for SO_DEBUG is for debugging to be turned off.

The SO_DONTROUTE option requests that outgoing messages bypass the standard routing facilities. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. It is protocol-specific whether this option has any effect and how the outgoing network interface is chosen. Support for this option with each protocol is implementation-defined.

The SO_ERROR option is used only on *getsockopt()*. When this option is specified, *getsockopt()*

shall return any pending error on the socket and clear the error status. It shall return a value of 0 if there is no pending error. `SO_ERROR` may be used to check for asynchronous errors on connected connectionless-mode sockets or for other types of asynchronous errors. `SO_ERROR` has no default value.

The `SO_KEEPALIVE` option enables the periodic transmission of messages on a connected socket. The behavior of this option is protocol-specific. On a connection-mode socket for which a connection has been established, if `SO_KEEPALIVE` is enabled and the connected socket fails to respond to the keep-alive messages, the connection shall be broken. The default value for `SO_KEEPALIVE` is zero, specifying that this capability is turned off.

The `SO_LINGER` option controls the action of the interface when unsent messages are queued on a socket and a `close()` is performed. The details of this option are protocol-specific. If `SO_LINGER` is enabled, the system shall block the calling thread during `close()` until it can transmit the data or until the end of the interval indicated by the `l_linger` member, whichever comes first. If `SO_LINGER` is not specified, and `close()` is issued, the system handles the call in a way that allows the calling thread to continue as quickly as possible. The default value for `SO_LINGER` is zero, or off, for the `l_onoff` element of the option value and zero seconds for the linger time specified by the `l_linger` element.

The `SO_OOBINLINE` option is valid only on protocols that support out-of-band data. The `SO_OOBINLINE` option requests that out-of-band data be placed in the normal data input queue as received; it is then accessible using the `read()` or `recv()` functions without the `MSG_OOB` flag set. The default for `SO_OOBINLINE` is off; that is, for out-of-band data not to be placed in the normal data input queue.

The `SO_RCVBUF` option requests that the buffer space allocated for receive operations on this socket be set to the value, in bytes, of the option value. Applications may wish to increase buffer size for high volume connections, or may decrease buffer size to limit the possible backlog of incoming data. The default value for the `SO_RCVBUF` option value is implementation-defined, and may vary by protocol.

The `SO_RCVLOWAT` option sets the minimum number of bytes to process for socket input operations. In general, receive calls block until any (non-zero) amount of data is received, then return the smaller of the amount available or the amount requested. The default value for `SO_RCVLOWAT` is 1, and does not affect the general case. If `SO_RCVLOWAT` is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. Receive calls may still return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned (for example, out-of-band data). As mentioned previously, the default value for `SO_RCVLOWAT` is 1 byte. It is implementation-defined whether the `SO_RCVLOWAT` option can be set.

The `SO_RCVTIMEO` option is an option to set a timeout value for input operations. It accepts a `timeval` structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or `errno` shall be set to `[EAGAIN]` or `[EWOULDBLOCK]` if no data were received. The default for this option is the value zero, which indicates that a receive operation will not time out. It is implementation-defined whether the `SO_RCVTIMEO` option can be set.

The `SO_REUSEADDR` option indicates that the rules used in validating addresses supplied in a `bind()` should allow reuse of local addresses. Operation of this option is protocol-specific. The default value for `SO_REUSEADDR` is off; that is, reuse of local addresses is not permitted.

The `SO_SNDBUF` option requests that the buffer space allocated for send operations on this socket be set to the value, in bytes, of the option value. The default value for the `SO_SNDBUF`

option value is implementation-defined, and may vary by protocol.

The `SO_SNDLOWAT` option sets the minimum number of bytes to process for socket output operations. Most output operations process all of the data supplied by the call, delivering data to the protocol for transmission and blocking as necessary for flow control. Non-blocking output operations process as much data as permitted subject to flow control without blocking, but process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. A `select()` operation testing the ability to write to a socket shall return true only if the send low water mark could be processed. The default value for `SO_SNDLOWAT` is implementation-defined and protocol-specific. It is implementation-defined whether the `SO_SNDLOWAT` option can be set.

The `SO_SNDTIMEO` option is an option to set a timeout value for the amount of time that an output function shall block because flow control prevents data from being sent. As noted in Table 2-2 (on page 523), the option value is a `timeval` structure with the number of seconds and microseconds specifying the limit on how long to wait for an output operation to complete. If a send operation has blocked for this much time, it shall return with a partial count or `errno` set to `[EAGAIN]` or `[EWOULDBLOCK]` if no data were sent. The default for this option is the value zero, which indicates that a send operation will not time out. It is implementation-defined whether the `SO_SNDTIMEO` option can be set.

The `SO_TYPE` option is used only on `getsockopt()`. When this option is specified, `getsockopt()` shall return the type of the socket (for example, `SOCK_STREAM`). This option is useful to servers that inherit sockets on start-up. `SO_TYPE` has no default value.

2.10.17 Use of Sockets for Local UNIX Connections

Support for UNIX domain sockets is mandatory.

UNIX domain sockets provide process-to-process communication in a single system.

2.10.17.1 Headers

The symbolic constant `AF_UNIX` defined in the `<sys/socket.h>` header is used to identify the UNIX domain address family. The `<sys/un.h>` header contains other definitions used in connection with UNIX domain sockets. See XBD Chapter 13 (on page 219).

The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to accommodate a `sockaddr_un` structure (see the `<sys/un.h>` header defined in XBD Chapter 13, on page 219) and shall be aligned at an appropriate boundary so that pointers to it can be cast as pointers to `sockaddr_un` structures and used to access the fields of those structures without alignment problems. When a `sockaddr_storage` structure is cast as a `sockaddr_un` structure, the `ss_family` field maps onto the `sun_family` field.

2.10.18 Use of Sockets over Internet Protocols

When a socket is created in the Internet family with a protocol value of zero, the implementation shall use the protocol listed below for the type of socket created.

`SOCK_STREAM` `IPPROTO_TCP`.

`SOCK_DGRAM` `IPPROTO_UDP`.

18253	RS	SOCK_RAW	IPPROTO_RAW.
18254		SOCK_SEQPACKET	Unspecified.
18255	RS	A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The default protocol for type SOCK_RAW shall be identified in the IP header with the value IPPROTO_RAW. Applications should not use the default protocol when creating a socket with type SOCK_RAW, but should identify a specific protocol by value. The ICMP control protocol is accessible from a raw socket by specifying a value of IPPROTO_ICMP for protocol.	
18256			
18257			
18258			
18259			
18260		2.10.19 Use of Sockets over Internet Protocols Based on IPv4	
18261		Support for sockets over Internet protocols based on IPv4 is mandatory.	
18262		2.10.19.1 Headers	
18263		The symbolic constant AF_INET defined in the <code><sys/socket.h></code> header is used to identify the IPv4 Internet address family. The <code><netinet/in.h></code> header contains other definitions used in connection with IPv4 Internet sockets. See XBD Chapter 13 (on page 219).	
18264			
18265			
18266		The <code>sockaddr_storage</code> structure defined in <code><sys/socket.h></code> shall be large enough to accommodate a <code>sockaddr_in</code> structure (see the <code><netinet/in.h></code> header defined in XBD Chapter 13, on page 219) and shall be aligned at an appropriate boundary so that pointers to it can be cast as pointers to <code>sockaddr_in</code> structures and used to access the fields of those structures without alignment problems. When a <code>sockaddr_storage</code> structure is cast as a <code>sockaddr_in</code> structure, the <code>ss_family</code> field maps onto the <code>sin_family</code> field.	
18267			
18268			
18269			
18270			
18271			
18272		2.10.20 Use of Sockets over Internet Protocols Based on IPv6	
18273	IP6	This section describes extensions to support sockets over Internet protocols based on IPv6. The functionality described in this section shall be provided on implementations that support the IPV6 option (and the rest of this section is not further shaded for this option).	
18274			
18275			
18276		To enable smooth transition from IPv4 to IPv6, the features defined in this section may, in certain circumstances, also be used in connection with IPv4; see Section 2.10.20.2 (on page 527).	
18277			
18278		2.10.20.1 Addressing	
18279		IPv6 overcomes the addressing limitations of earlier versions by using 128-bit addresses instead of 32-bit addresses. The IPv6 address architecture is described in RFC 2373.	
18280			
18281		There are three kinds of IPv6 address:	
18282		Unicast	
18283		Identifies a single interface.	
18284		A unicast address can be global, link-local (designed for use on a single link), or site-local (designed for systems not connected to the Internet). Link-local and site-local addresses need not be globally unique.	
18285			
18286			
18287		Anycast	
18288		Identifies a set of interfaces such that a packet sent to the address can be delivered to any member of the set.	
18289			
18290		An anycast address is similar to a unicast address; the nodes to which an anycast address is	

General Information

Sockets

- 18291 assigned must be explicitly configured to know that it is an anycast address.
- 18292 Multicast
- 18293 Identifies a set of interfaces such that a packet sent to the address should be delivered to
- 18294 every member of the set.
- 18295 An application can send multicast datagrams by simply specifying an IPv6 multicast
- 18296 address in the *address* argument of *sendto()*. To receive multicast datagrams, an application
- 18297 must join the multicast group (using *setsockopt()* with *IPV6_JOIN_GROUP*) and must bind
- 18298 to the socket the UDP port on which datagrams will be received. Some applications should
- 18299 also bind the multicast group address to the socket, to prevent other datagrams destined to
- 18300 that port from being delivered to the socket.
- 18301 A multicast address can be global, node-local, link-local, site-local, or organization-local.
- 18302 The following special IPv6 addresses are defined:
- 18303 Unspecified
- 18304 An address that is not assigned to any interface and is used to indicate the absence of an
- 18305 address.
- 18306 Loopback
- 18307 A unicast address that is not assigned to any interface and can be used by a node to send
- 18308 packets to itself.
- 18309 Two sets of IPv6 addresses are defined to correspond to IPv4 addresses:
- 18310 IPv4-compatible addresses
- 18311 These are assigned to nodes that support IPv6 and can be used when traffic is “tunneled”
- 18312 through IPv4.
- 18313 IPv4-mapped addresses
- 18314 These are used to represent IPv4 addresses in IPv6 address format; see [Section 2.10.20.2](#).
- 18315 Note that the unspecified address and the loopback address must not be treated as
- 18316 IPv4-compatible addresses.
- 18317 2.10.20.2 *Compatibility with IPv4*
- 18318 The API provides the ability for IPv6 applications to interoperate with applications using IPv4,
- 18319 by using IPv4-mapped IPv6 addresses. These addresses can be generated automatically by the
- 18320 *getaddrinfo()* function when the specified host has only IPv4 addresses.
- 18321 Applications can use *AF_INET6* sockets to open TCP connections to IPv4 nodes, or send UDP
- 18322 packets to IPv4 nodes, by simply encoding the destination’s IPv4 address as an IPv4-mapped
- 18323 IPv6 address, and passing that address, within a **sockaddr_in6** structure, in the *connect()*,
- 18324 *sendto()*, or *sendmsg()* function. When applications use *AF_INET6* sockets to accept TCP
- 18325 connections from IPv4 nodes, or receive UDP packets from IPv4 nodes, the system shall return
- 18326 the peer’s address to the application in the *accept()*, *recvfrom()*, *recvmsg()*, or *getpeername()*
- 18327 function using a **sockaddr_in6** structure encoded this way. If a node has an IPv4 address, then
- 18328 the implementation shall allow applications to communicate using that address via an
- 18329 *AF_INET6* socket. In such a case, the address will be represented at the API by the
- 18330 corresponding IPv4-mapped IPv6 address. Also, the implementation may allow an *AF_INET6*
- 18331 socket bound to **in6addr_any** to receive inbound connections and packets destined to one of the
- 18332 node’s IPv4 addresses.
- 18333 An application can use *AF_INET6* sockets to bind to a node’s IPv4 address by specifying the
- 18334 address as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure in the *bind()* function. For
- 18335 an *AF_INET6* socket bound to a node’s IPv4 address, the system shall return the address in the

18336 *getsockname()* function as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure.

18337 2.10.20.3 Interface Identification

18338 Each local interface is assigned a unique positive integer as a numeric index. Indexes start at 1;
 18339 zero is not used. There may be gaps so that there is no current interface for a particular positive
 18340 index. Each interface also has a unique implementation-defined name.

18341 2.10.20.4 Options

18342 The following options apply at the IPPROTO_IPV6 level:

18343 IPV6_JOIN_GROUP

18344 When set via *setsockopt()*, it joins the application to a multicast group on an interface
 18345 (identified by its index) and addressed by a given multicast address, enabling packets sent
 18346 to that address to be read via the socket. If the interface index is specified as zero, the
 18347 system selects the interface (for example, by looking up the address in a routing table and
 18348 using the resulting interface).

18349 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

18350 The parameter type of this option is a pointer to an **ipv6_mreq** structure.

18351 IPV6_LEAVE_GROUP

18352 When set via *setsockopt()*, it removes the application from the multicast group on an
 18353 interface (identified by its index) and addressed by a given multicast address.

18354 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

18355 The parameter type of this option is a pointer to an **ipv6_mreq** structure.

18356 IPV6_MULTICAST_HOPS

18357 The value of this option is the hop limit for outgoing multicast IPv6 packets sent via the
 18358 socket. Its possible values are the same as those of IPV6_UNICAST_HOPS. If the
 18359 IPV6_MULTICAST_HOPS option is not set, a value of 1 is assumed. This option can be set
 18360 via *setsockopt()* and read via *getsockopt()*.

18361 The parameter type of this option is a pointer to an **int**. (Default value: 1)

18362 IPV6_MULTICAST_IF

18363 The index of the interface to be used for outgoing multicast packets. It can be set via
 18364 *setsockopt()* and read via *getsockopt()*. If the interface index is specified as zero, the system
 18365 selects the interface (for example, by looking up the address in a routing table and using the
 18366 resulting interface).

18367 The parameter type of this option is a pointer to an **unsigned int**. (Default value: 0)

18368 IPV6_MULTICAST_LOOP

18369 This option controls whether outgoing multicast packets should be delivered back to the
 18370 local application when the sending interface is itself a member of the destination multicast
 18371 group. If it is set to 1 they are delivered. If it is set to 0 they are not. Other values result in an
 18372 [EINVAL] error. This option can be set via *setsockopt()* and read via *getsockopt()*.

18373 The parameter type of this option is a pointer to an **unsigned int** which is used as a Boolean
 18374 value. (Default value: 1)

18375 IPV6_UNICAST_HOPS

18376 The value of this option is the hop limit for outgoing unicast IPv6 packets sent via the
 18377 socket. If the option is not set, or is set to -1, the system selects a default value. Attempts to

set a value less than `-1` or greater than `255` shall result in an `[EINVAL]` error. This option can be set via `setsockopt()` and read via `getsockopt()`.

The parameter type of this option is a pointer to an **int**. (Default value: Unspecified)

IPV6_V6ONLY

This socket option restricts `AF_INET6` sockets to IPv6 communications only. `AF_INET6` sockets may be used for both IPv4 and IPv6 communications. Some applications may want to restrict their use of an `AF_INET6` socket to IPv6 communications only. For these applications, the `IPV6_V6ONLY` socket option is defined. When this option is turned on, the socket can be used to send and receive IPv6 packets only. This is an `IPPROTO_IPV6`-level option.

The parameter type of this option is a pointer to an **int** which is used as a Boolean value. (Default value: 0)

An `[EOPNOTSUPP]` error shall result if `IPV6_JOIN_GROUP` or `IPV6_LEAVE_GROUP` is used with `getsockopt()`.

2.10.20.5 Headers

The symbolic constant `AF_INET6` is defined in the `<sys/socket.h>` header to identify the IPv6 Internet address family. See XBD Chapter 13 (on page 219).

The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to accommodate a `sockaddr_in6` structure (see the `<netinet/in.h>` header defined in XBD Chapter 13, on page 219) and shall be aligned at an appropriate boundary so that pointers to it can be cast as pointers to `sockaddr_in6` structures and used to access the fields of those structures without alignment problems. When a `sockaddr_storage` structure is cast as a `sockaddr_in6` structure, the `ss_family` field maps onto the `sin6_family` field.

The `<netinet/in.h>`, `<arpa/inet.h>`, and `<netdb.h>` headers contain other definitions used in connection with IPv6 Internet sockets; see XBD Chapter 13 (on page 219).

2.11 Tracing

OB TRC

This section describes extensions to support tracing of user applications. The functionality described in this section is dependent on support of the Trace option (and the rest of this section is not further shaded for this option).

The tracing facilities defined in POSIX.1-2008 allow a process to select a set of trace event types, to activate a trace stream of the selected trace events as they occur in the flow of execution, and to retrieve the recorded trace events.

The tracing operation relies on three logically different components: the traced process, the controller process, and the analyzer process. During the execution of the traced process, when a trace point is reached, a trace event is recorded into the trace streams created for that process in which the associated trace event type identifier is not being filtered out. The controller process controls the operation of recording the trace events into the trace stream. It shall be able to:

- Initialize the attributes of a trace stream
- Create the trace stream (for a specified traced process) using those attributes

- Start and stop tracing for the trace stream
- Filter the type of trace events to be recorded, if the Trace Event Filter option is supported
- Shut a trace stream down

These operations can be done for an active trace stream. The analyzer process retrieves the traced events either at runtime, when the trace stream has not yet been shut down, but is still recording trace events; or after opening a trace log that had been previously recorded and shut down. These three logically different operations can be performed by the same process, or can be distributed into different processes.

A trace stream identifier can be created by a call to `posix_trace_create()`, `posix_trace_create_withlog()`, or `posix_trace_open()`. The `posix_trace_create()` and `posix_trace_create_withlog()` functions should be used by a controller process. The `posix_trace_open()` should be used by an analyzer process.

The tracing functions can serve different purposes. One purpose is debugging the possibly pre-instrumented code, while another is post-mortem fault analysis. These two potential uses differ in that the first requires pre-filtering capabilities to avoid overwhelming the trace stream and permits focusing on expected information; while the second needs comprehensive trace capabilities in order to be able to record all types of information.

The events to be traced belong to two classes:

1. User trace events (generated by the application instrumentation)
2. System trace events (generated by the operating system)

The trace interface defines several system trace event types associated with control of and operation of the trace stream. This small set of system trace events includes the minimum required to interpret correctly the trace event information present in the stream. Other desirable system trace events for some particular application profile may be implemented and are encouraged; for example, process and thread scheduling, signal occurrence, and so on.

Each traced process shall have a mapping of the trace event names to trace event type identifiers that have been defined for that process. Each active trace stream shall have a mapping that incorporates all the trace event type identifiers predefined by the trace system plus all the mappings of trace event names to trace event type identifiers of the processes that are being traced into that trace stream. These mappings are defined from the instrumented application by calling the `posix_trace_eventid_open()` function and from the controller process by calling the `posix_trace_trid_eventid_open()` function. For a pre-recorded trace stream, the list of trace event types is obtained from the pre-recorded trace log.

The last data modification and file status change timestamps of a file associated with an active trace stream shall be marked for update every time any of the tracing operations modifies that file.

The last data access timestamp of a file associated with a trace stream shall be marked for update every time any of the tracing operations causes data to be read from that file.

Results are undefined if the application performs any operation on a file descriptor associated with an active or pre-recorded trace stream until `posix_trace_shutdown()` or `posix_trace_close()` is called for that trace stream. Results are also undefined if the analyzer process and the traced process do not share the same programming environment (see *c99*, Programming Environments in the Shell and Utilities volume of POSIX.1-2008).

The main purpose of this option is to define a complete set of functions and concepts that allow a conforming application to be traced from creation to termination, whatever its realtime constraints and properties.

2.11.1 Tracing Data Definitions

2.11.1.1 Structures

The **<trace.h>** header shall define the *posix_trace_status_info* and *posix_trace_event_info* structures described below. Implementations may add extensions to these structures.

posix_trace_status_info Structure

To facilitate control of a trace stream, information about the current state of an active trace stream can be obtained dynamically. This structure is returned by a call to the *posix_trace_get_status()* function.

The **posix_trace_status_info** structure defined in **<trace.h>** shall contain at least the following members:

Member Type	Member Name	Description
int	<i>posix_stream_status</i>	The operating mode of the trace stream.
int	<i>posix_stream_full_status</i>	The full status of the trace stream.
int	<i>posix_stream_overrun_status</i>	Indicates whether trace events were lost in the trace stream.

If the Trace Log option is supported in addition to the Trace option, the **posix_trace_status_info** structure defined in **<trace.h>** shall contain at least the following additional members:

Member Type	Member Name	Description
int	<i>posix_stream_flush_status</i>	Indicates whether a flush is in progress.
int	<i>posix_stream_flush_error</i>	Indicates whether any error occurred during the last flush operation.
int	<i>posix_log_overrun_status</i>	Indicates whether trace events were lost in the trace log.
int	<i>posix_log_full_status</i>	The full status of the trace log.

The *posix_stream_status* member indicates the operating mode of the trace stream and shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_RUNNING

Tracing is in progress; that is, the trace stream is accepting trace events.

POSIX_TRACE_SUSPENDED

The trace stream is not accepting trace events. The tracing operation has not yet started or has stopped, either following a *posix_trace_stop()* function call or because the trace resources are exhausted.

The *posix_stream_full_status* member indicates the full status of the trace stream, and it shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_FULL

The space in the trace stream for trace events is exhausted.

POSIX_TRACE_NOT_FULL

There is still space available in the trace stream.

The combination of the *posix_stream_status* and *posix_stream_full_status* members also indicates the actual status of the stream. The status shall be interpreted as follows:

- 18503 POSIX_TRACE_RUNNING and POSIX_TRACE_NOT_FULL
 18504 This status combination indicates that tracing is in progress, and there is space available for
 18505 recording more trace events.
- 18506 POSIX_TRACE_RUNNING and POSIX_TRACE_FULL
 18507 This status combination indicates that tracing is in progress and that the trace stream is full
 18508 of trace events. This status combination cannot occur unless the *stream-full-policy* is set to
 18509 POSIX_TRACE_LOOP. The trace stream contains trace events recorded during a moving
 18510 time window of prior trace events, and some older trace events may have been overwritten
 18511 and thus lost.
- 18512 POSIX_TRACE_SUSPENDED and POSIX_TRACE_NOT_FULL
 18513 This status combination indicates that tracing has not yet been started, has been stopped by
 18514 the *posix_trace_stop()* function, or has been cleared by the *posix_trace_clear()* function.
- 18515 POSIX_TRACE_SUSPENDED and POSIX_TRACE_FULL
 18516 This status combination indicates that tracing has been stopped by the implementation
 18517 because the *stream-full-policy* attribute was POSIX_TRACE_UNTIL_FULL and trace
 18518 resources were exhausted, or that the trace stream was stopped by the function
 18519 *posix_trace_stop()* at a time when trace resources were exhausted.
- 18520 The *posix_stream_overrun_status* member indicates whether trace events were lost in the trace
 18521 stream, and shall have one of the following values defined by manifest constants in the
 18522 **<trace.h>** header:
- 18523 POSIX_TRACE_OVERRUN
 18524 At least one trace event was lost and thus was not recorded in the trace stream.
- 18525 POSIX_TRACE_NO_OVERRUN
 18526 No trace events were lost.
- 18527 When the corresponding trace stream is created, the *posix_stream_overrun_status* member shall be
 18528 set to POSIX_TRACE_NO_OVERRUN.
- 18529 Whenever an overrun occurs, the *posix_stream_overrun_status* member shall be set to
 18530 POSIX_TRACE_OVERRUN.
- 18531 An overrun occurs when:
- 18532 • The policy is POSIX_TRACE_LOOP and a recorded trace event is overwritten.
 - 18533 • The policy is POSIX_TRACE_UNTIL_FULL and the trace stream is full when a trace event
 18534 is generated.
 - 18535 • If the Trace Log option is supported, the policy is POSIX_TRACE_FLUSH and at least one
 18536 trace event is lost while flushing the trace stream to the trace log.
- 18537 The *posix_stream_overrun_status* member is reset to zero after its value is read.
- 18538 If the Trace Log option is supported in addition to the Trace option, the *posix_stream_flush_status*,
 18539 *posix_stream_flush_error*, *posix_log_overrun_status*, and *posix_log_full_status* members are defined
 18540 as follows; otherwise, they are undefined.
- 18541 The *posix_stream_flush_status* member indicates whether a flush operation is being performed
 18542 and shall have one of the following values defined by manifest constants in the header
 18543 **<trace.h>**:
- 18544 POSIX_TRACE_FLUSHING
 18545 The trace stream is currently being flushed to the trace log.

POSIX_TRACE_NOT_FLUSHING

No flush operation is in progress.

The *posix_stream_flush_status* member shall be set to POSIX_TRACE_FLUSHING if a flush operation is in progress either due to a call to the *posix_trace_flush()* function (explicit or caused by a trace stream shutdown operation) or because the trace stream has become full with the *stream-full-policy* attribute set to POSIX_TRACE_FLUSH. The *posix_stream_flush_status* member shall be set to POSIX_TRACE_NOT_FLUSHING if no flush operation is in progress.

The *posix_stream_flush_error* member shall be set to zero if no error occurred during flushing. If an error occurred during a previous flushing operation, the *posix_stream_flush_error* member shall be set to the value of the first error that occurred. If more than one error occurs while flushing, error values after the first shall be discarded. The *posix_stream_flush_error* member is reset to zero after its value is read.

The *posix_log_ouerrun_status* member indicates whether trace events were lost in the trace log, and shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_OVERRUN

At least one trace event was lost.

POSIX_TRACE_NO_OVERRUN

No trace events were lost.

When the corresponding trace stream is created, the *posix_log_ouerrun_status* member shall be set to POSIX_TRACE_NO_OVERRUN. Whenever an overrun occurs, this status shall be set to POSIX_TRACE_OVERRUN. The *posix_log_ouerrun_status* member is reset to zero after its value is read.

The *posix_log_full_status* member indicates the full status of the trace log, and it shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_FULL

The space in the trace log is exhausted.

POSIX_TRACE_NOT_FULL

There is still space available in the trace log.

The *posix_log_full_status* member is only meaningful if the *log-full-policy* attribute is either POSIX_TRACE_UNTIL_FULL or POSIX_TRACE_LOOP.

For an active trace stream without log, that is created by the *posix_trace_create()* function, the *posix_log_ouerrun_status* member shall be set to POSIX_TRACE_NO_OVERRUN and the *posix_log_full_status* member shall be set to POSIX_TRACE_NOT_FULL.

posix_trace_event_info Structure

The trace event structure **posix_trace_event_info** contains the information for one recorded trace event. This structure is returned by the set of functions *posix_trace_getnext_event()*, *posix_trace_timedgetnext_event()*, and *posix_trace_trygetnext_event()*.

The **posix_trace_event_info** structure defined in **<trace.h>** shall contain at least the following members:

Member Type	Member Name	Description
trace_event_id_t	<i>posix_event_id</i>	Trace event type identification.
pid_t	<i>posix_pid</i>	Process ID of the process that generated the trace event.
void *	<i>posix_prog_address</i>	Address at which the trace point was invoked.
int	<i>posix_truncation_status</i>	Status about the truncation of the data associated with this trace event.
struct timespec	<i>posix_timestamp</i>	Time at which the trace event was generated.

In addition, the **posix_trace_event_info** structure defined in **<trace.h>** shall contain the following additional member:

Member Type	Member Name	Description
pthread_t	<i>posix_thread_id</i>	Thread ID of the thread that generated the trace event.

The *posix_event_id* member represents the identification of the trace event type and its value is not directly defined by the user. This identification is returned by a call to one of the following functions: *posix_trace_trid_eventid_open()*, *posix_trace_eventtypelist_getnext_id()*, or *posix_trace_eventid_open()*. The name of the trace event type can be obtained by calling *posix_trace_eventid_get_name()*.

The *posix_pid* is the process identifier of the traced process which generated the trace event. If the *posix_event_id* member is one of the implementation-defined system trace events and that trace event is not associated with any process, the *posix_pid* member shall be set to zero.

For a user trace event, the *posix_prog_address* member is the process mapped address of the point at which the associated call to the *posix_trace_event()* function was made. For a system trace event, if the trace event is caused by a system service explicitly called by the application, the *posix_prog_address* member shall be the address of the process at the point where the call to that system service was made.

The *posix_truncation_status* member defines whether the data associated with a trace event has been truncated at the time the trace event was generated, or at the time the trace event was read from the trace stream, or (if the Trace Log option is supported) from the trace log (see the *event* argument from the *posix_trace_getnext_event()* function). The *posix_truncation_status* member shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_NOT_TRUNCATED

All the traced data is available.

POSIX_TRACE_TRUNCATED_RECORD

Data was truncated at the time the trace event was generated.

POSIX_TRACE_TRUNCATED_READ

Data was truncated at the time the trace event was read from a trace stream or a trace log because the reader's buffer was too small. This truncation status overrides the **POSIX_TRACE_TRUNCATED_RECORD** status.

The *posix_timestamp* member shall be the time at which the trace event was generated. The clock used is implementation-defined, but the resolution of this clock can be retrieved by a call to the *posix_trace_attr_getclockres()* function.

The *posix_thread_id* member is the identifier of the thread that generated the trace event. If the *posix_event_id* member is one of the implementation-defined system trace events and that trace

event is not associated with any thread, the *posix_thread_id* member shall be set to zero.

2.11.1.2 Trace Stream Attributes

Trace streams have attributes that compose the **posix_trace_attr_t** trace stream attributes object. This object shall contain at least the following attributes:

- The *generation-version* attribute identifies the origin and version of the trace system.
- The *trace-name* attribute is a character string defined by the trace controller, and that identifies the trace stream.
- The *creation-time* attribute represents the time of the creation of the trace stream.
- The *clock-resolution* attribute defines the clock resolution of the clock used to generate timestamps.
- The *stream-min-size* attribute defines the minimum size in bytes of the trace stream strictly reserved for the trace events.
- The *stream-full-policy* attribute defines the policy followed when the trace stream is full; its value is `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or `POSIX_TRACE_FLUSH`.
- The *max-data-size* attribute defines the maximum record size in bytes of a trace event.

In addition, if the Trace option and the Trace Inherit option are both supported, the **posix_trace_attr_t** trace stream creation attributes object shall contain at least the following attributes:

- The *inheritance* attribute specifies whether a newly created trace stream will inherit tracing in its parent's process trace stream. It is either `POSIX_TRACE_INHERITED` or `POSIX_TRACE_CLOSE_FOR_CHILD`.

In addition, if the Trace option and the Trace Log option are both supported, the **posix_trace_attr_t** trace stream creation attributes object shall contain at least the following attribute:

- If the file type corresponding to the trace log supports the `POSIX_TRACE_LOOP` or the `POSIX_TRACE_UNTIL_FULL` policies, the *log-max-size* attribute defines the maximum size in bytes of the trace log associated with an active trace stream. Other stream data—for example, trace attribute values—shall not be included in this size.
- The *log-full-policy* attribute defines the policy of a trace log associated with an active trace stream to be `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or `POSIX_TRACE_APPEND`.

2.11.2 Trace Event Type Definitions

2.11.2.1 System Trace Event Type Definitions

The following system trace event types, defined in the **<trace.h>** header, track the invocation of the trace operations:

- `POSIX_TRACE_START` shall be associated with a trace start operation.

- 18668
- POSIX_TRACE_STOP shall be associated with a trace stop operation.
- 18669
- If the Trace Event Filter option is supported, POSIX_TRACE_FILTER shall be associated
- 18670
- with a trace event type filter change operation.
- 18671
- The following system trace event types, defined in the `<trace.h>` header, report operational trace
- 18672
- events:
- 18673
- POSIX_TRACE_OVERFLOW shall mark the beginning of a trace overflow condition.
- 18674
- POSIX_TRACE_RESUME shall mark the end of a trace overflow condition.
- 18675
- If the Trace Log option is supported, POSIX_TRACE_FLUSH_START shall mark the
- 18676
- beginning of a flush operation.
- 18677
- If the Trace Log option is supported, POSIX_TRACE_FLUSH_STOP shall mark the end of
- 18678
- a flush operation.
- 18679
- If an implementation-defined trace error condition is reported, it shall be marked
- 18680
- POSIX_TRACE_ERROR.
- 18681
- The interpretation of a trace stream or a trace log by a trace analyzer process relies on the
- 18682
- information recorded for each trace event, and also on system trace events that indicate the
- 18683
- invocation of trace control operations and trace system operational trace events.
- 18684
- The POSIX_TRACE_START and POSIX_TRACE_STOP trace events specify the time windows
- 18685
- during which the trace stream is running.
- 18686
- The POSIX_TRACE_STOP trace event with an associated data that is equal to zero
- 18687
- indicates a call of the function `posix_trace_stop()`.
- 18688
- The POSIX_TRACE_STOP trace event with an associated data that is different from zero
- 18689
- indicates an automatic stop of the trace stream (see the definition of the
- 18690
- `posix_trace_attr_getstreamfullpolicy()` function in `posix_trace_attr_getinherited()`).
- 18691
- The POSIX_TRACE_FILTER trace event indicates that a trace event type filter value changed
- 18692
- while the trace stream was running.
- 18693
- The POSIX_TRACE_ERROR serves to inform the analyzer process that an implementation-
- 18694
- defined internal error of the trace system occurred.
- 18695
- The POSIX_TRACE_OVERFLOW trace event shall be reported with a timestamp equal to the
- 18696
- timestamp of the first trace event overwritten. This is an indication that some generated trace
- 18697
- events have been lost.
- 18698
- The POSIX_TRACE_RESUME trace event shall be reported with a timestamp equal to the
- 18699
- timestamp of the first valid trace event reported after the overflow condition ends and shall be
- 18700
- reported before this first valid trace event. This is an indication that the trace system is reliably
- 18701
- recording trace events after an overflow condition.
- 18702
- Each of these trace event types shall be defined by a constant trace event name and a
- 18703
- trace_event_id_t** constant; trace event data is associated with some of these trace events.
- 18704
- If the Trace option is supported and the Trace Event Filter option and the Trace Log option are
- 18705
- not supported, the following predefined system trace events in Table 2-3 (on page 537) shall be
- 18706
- defined:

Table 2-3 Trace Option: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

If the Trace option and the Trace Event Filter option are both supported, and if the Trace Log option is not supported, the following predefined system trace events in Table 2-4 shall be defined:

Table 2-4 Trace and Trace Event Filter Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	event_filter
		trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter
		new_event_filter
		trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

If the Trace option and the Trace Log option are both supported, and if the Trace Event Filter option is not supported, the following predefined system trace events in Table 2-5 (on page 538) shall be defined:

Table 2-5 Trace and Trace Log Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

If the Trace option, the Trace Event Filter option, and the Trace Log option are all supported, the following predefined system trace events in [Table 2-6](#) shall be defined:

Table 2-6 Trace, Trace Log, and Trace Event Filter Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error int
posix_trace_start	POSIX_TRACE_START	event_filter trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter new_event_filter trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

2.11.2.2 User Trace Event Type Definitions

The user trace event `POSIX_TRACE_UNNAMED_USEREVENT` is defined in the `<trace.h>` header. If the limit of per-process user trace event names represented by `{TRACE_USER_EVENT_MAX}` has already been reached, this predefined user event shall be returned when the application tries to register more events than allowed. The data associated with this trace event is application-defined.

The following predefined user trace event in [Table 2-7](#) (on page 539) shall be defined:

Table 2-7 Trace Option: User Trace Event

Event Name	Constant
posix_trace_unnamed_userevent	POSIX_TRACE_UNNAMED_USEREVENT

2.11.3 Trace Functions

The trace interface is built and structured to improve portability through use of trace data of opaque type. The object-oriented approach for the manipulation of trace attributes and trace event type identifiers requires definition of many constructor and selector functions which operate on these opaque types. Also, the trace interface must support several different tracing roles. To facilitate reading the trace interface, the trace functions are grouped into small functional sets supporting the three different roles:

- A trace controller process requires functions to set up and customize all the resources needed to run a trace stream, including:
 - Attribute initialization and destruction (*posix_trace_attr_init()*)
 - Identification information manipulation (*posix_trace_attr_getgenversion()*)
 - Trace system behavior modification (*posix_trace_attr_getinherited()*)
 - Trace stream and trace log size set (*posix_trace_attr_getmaxusereventsize()*)
 - Trace stream creation, flush, and shutdown (*posix_trace_create()*)
 - Trace stream and trace log clear (*posix_trace_clear()*)
 - Trace event type identifier manipulation (*posix_trace_trid_eventid_open()*)
 - Trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id()*)
 - Trace event type set manipulation (*posix_trace_eventset_empty()*)
 - Trace event type filter set (*posix_trace_set_filter()*)
 - Trace stream start and stop (*posix_trace_start()*)
 - Trace stream information and status read (*posix_trace_get_attr()*)
- A traced process requires functions to instrument trace points:
 - Trace event type identifiers definition and trace points insertion (*posix_trace_event()*)
- A trace analyzer process requires functions to retrieve information from a trace stream and trace log:
 - Identification information read (*posix_trace_attr_getgenversion()*)
 - Trace system behavior information read (*posix_trace_attr_getinherited()*)
 - Trace stream and trace log size get (*posix_trace_attr_getmaxusereventsize()*)
 - Trace event type identifier manipulation (*posix_trace_trid_eventid_open()*)
 - Trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id()*)
 - Trace log open, rewind, and close (*posix_trace_open()*)

- 18808 — Trace stream information and status read (*posix_trace_get_attr()*)
- 18809 — Trace event read (*posix_trace_getnext_event()*)

18810 2.12 Data Types

18811 2.12.1 Defined Types

18812 All of the data types used by various functions are defined by the implementation. The
 18813 following table describes some of these types. Other types referenced in the description of a
 18814 function, not mentioned here, can be found in the appropriate header for that function.

Defined Type	Description
cc_t	Type used for terminal special characters.
clock_t	Integer or real-floating type used for processor times, as defined in the ISO C standard.
clockid_t	Used for clock ID type in some timer functions.
dev_t	Arithmetic type used for device numbers.
DIR	Type representing a directory stream.
div_t	Structure type returned by the <i>div()</i> function.
FILE	Structure containing information about a file.
glob_t	Structure type used in pathname pattern matching.
fpos_t	Type containing all information needed to specify uniquely every position within a file.
gid_t	Integer type used for group IDs.
iconv_t	Type used for conversion descriptors.
id_t	Integer type used as a general identifier; can be used to contain at least the largest of a pid_t , uid_t , or gid_t .
ino_t	Unsigned integer type used for file serial numbers.
key_t	Arithmetic type used for XSI interprocess communication.
ldiv_t	Structure type returned by the <i>ldiv()</i> function.
mode_t	Integer type used for file attributes.
mqd_t	Used for message queue descriptors.
nfds_t	Integer type used for the number of file descriptors.
nlink_t	Integer type used for link counts.
off_t	Signed integer type used for file sizes.
pid_t	Signed integer type used for process and process group IDs.
pthread_attr_t	Used to identify a thread attribute object.
pthread_cond_t	Used for condition variables.
pthread_condattr_t	Used to identify a condition attribute object.
pthread_key_t	Used for thread-specific data keys.
pthread_mutex_t	Used for mutexes.
pthread_mutexattr_t	Used to identify a mutex attribute object.
pthread_once_t	Used for dynamic package initialization.
pthread_rwlock_t	Used for read-write locks.
pthread_rwlockattr_t	Used for read-write lock attributes.
pthread_t	Used to identify a thread.

	Defined Type	Description
18850		
18851	ptrdiff_t	Signed integer type of the result of subtracting two pointers.
18852	regex_t	Structure type used in regular expression matching.
18853	regmatch_t	Structure type used in regular expression matching.
18854	rlim_t	Unsigned integer type used for limit values, to which objects of type int and off_t can be cast without loss of value.
18855		
18856	sem_t	Type used in performing semaphore operations.
18857	sig_atomic_t	Integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.
18858		
18859	sigset_t	Integer or structure type of an object used to represent sets of signals.
18860		
18861	size_t	Unsigned integer type used for size of objects.
18862	speed_t	Type used for terminal baud rates.
18863	ssize_t	Signed integer type used for a count of bytes or an error indication.
18864		
18865	suseconds_t	Signed integer type used for time in microseconds.
18866	tcflag_t	Type used for terminal modes.
18867	time_t	Integer or real-floating type used for time in seconds, as defined in the ISO C standard.
18868		
18869	timer_t	Used for timer ID returned by the <i>timer_create()</i> function.
18870	uid_t	Integer type used for user IDs.
18871	va_list	Type used for traversing variable argument lists.
18872	wchar_t	Integer type whose range of values can represent distinct codes for all members of the largest extended character set specified by the supported locales.
18873		
18874		
18875	wctype_t	Scalar type which represents a character class descriptor.
18876	wint_t	Integer type capable of storing any valid value of wchar_t or WEOF.
18877		
18878	wordexp_t	Structure type used in word expansion.

18879 2.12.2 The char Type

18880 The type **char** is defined as a single byte; see XBD Chapter 3 (on page 33) (Byte and Character).

18881 2.12.3 Pointer Types

18882 All function pointer types shall have the same representation as the type pointer to **void**.
 18883 Conversion of a function pointer to **void *** shall not alter the representation. A **void *** value
 18884 resulting from such a conversion can be converted back to the original function pointer type,
 18885 using an explicit cast, without loss of information.

18886 **Note:** The ISO C standard does not require this, but it is required for POSIX conformance.

(blank page)

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

18887

Chapter 3

18888

System Interfaces

18889

18890

This chapter describes the functions, macros, and external variables to support applications portability at the C-language source level.

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

FD_CLR()*System Interfaces*18891 **NAME**

18892 FD_CLR — macros for synchronous I/O multiplexing

18893 **SYNOPSIS**

```
18894     #include <sys/select.h>
18895     void FD_CLR(int fd, fd_set *fdset);
18896     int FD_ISSET(int fd, fd_set *fdset);
18897     void FD_SET(int fd, fd_set *fdset);
18898     void FD_ZERO(fd_set *fdset);
```

18899 **DESCRIPTION**18900 Refer to *pselect()*.

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

System Interfaces

_Exit()**NAME**

_Exit, _exit — terminate a process

SYNOPSIS

```
#include <stdlib.h>
```

```
void _Exit(int status);
```

```
#include <unistd.h>
```

```
void _exit(int status);
```

DESCRIPTION

For **_Exit()**: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The value of *status* may be 0, EXIT_SUCCESS, EXIT_FAILURE, or any other value, though only the least significant 8 bits (that is, *status* & 0377) shall be available to a waiting parent process.

The **_Exit()** and **_exit()** functions shall be functionally equivalent.

The **_Exit()** and **_exit()** functions shall not call functions registered with **atexit()** nor any registered signal handlers. Open streams shall not be flushed. Whether open streams are closed (without flushing) is implementation-defined. Finally, the calling process shall be terminated with the consequences described below.

Consequences of Process Termination

Process termination caused by any reason shall have the following consequences:

Note: These consequences are all extensions to the ISO C standard and are not further CX shaded. However, functionality relating to the XSI option is shaded.

- All of the file descriptors, directory streams, conversion descriptors, and message catalog descriptors open in the calling process shall be closed.

- If the parent process of the calling process is executing a **wait()**, **waitid()**, or **waitpid()**, and has neither set its SA_NOCLDWAIT flag nor set SIGCHLD to SIG_IGN, it shall be notified of termination of the calling process and the low-order eight bits (that is, bits 0377) of *status* shall be made available to it. If the parent is not waiting, the child's status shall be made available to it when the parent subsequently executes **wait()**, **waitid()**, or **waitpid()**.

The semantics of the **waitid()** function shall be equivalent to **wait()**.

- If the parent process of the calling process is not executing a **wait()**, **waitid()**, or **waitpid()**, and has neither set its SA_NOCLDWAIT flag nor set SIGCHLD to SIG_IGN, the calling process shall be transformed into a *zombie process*. A *zombie process* is an inactive process and it shall be deleted at some later time when its parent process executes **wait()**, **waitid()**, or **waitpid()**.

The semantics of the **waitid()** function shall be equivalent to **wait()**.

- Termination of a process does not directly terminate its children. The sending of a SIGHUP signal as described below indirectly terminates children in some circumstances.

- Either:

If the implementation supports the SIGCHLD signal, a SIGCHLD shall be sent to the parent process.

Or:

_Exit()*System Interfaces*

- 18943 XSI If the parent process has set its SA_NOCLDWAIT flag, or set SIGCHLD to SIG_IGN, the
 18944 status shall be discarded, and the lifetime of the calling process shall end immediately. If
 18945 SA_NOCLDWAIT is set, it is implementation-defined whether a SIGCHLD signal is sent to
 18946 the parent process.
- 18947 • The parent process ID of all of the existing child processes and zombie processes of the
 18948 calling process shall be set to the process ID of an implementation-defined system process.
 18949 That is, these processes shall be inherited by a special system process.
- 18950 XSI • Each attached shared-memory segment is detached and the value of *shm_nattch* (see
 18951 *shmget()*) in the data structure associated with its shared memory ID shall be decremented
 18952 by 1.
- 18953 XSI • For each semaphore for which the calling process has set a *semadj* value (see *semop()*), that
 18954 value shall be added to the *semval* of the specified semaphore.
- 18955 • If the process is a controlling process, the SIGHUP signal shall be sent to each process in
 18956 the foreground process group of the controlling terminal belonging to the calling process.
- 18957 • If the process is a controlling process, the controlling terminal associated with the session
 18958 shall be disassociated from the session, allowing it to be acquired by a new controlling
 18959 process.
- 18960 • If the exit of the process causes a process group to become orphaned, and if any member of
 18961 the newly-orphaned process group is stopped, then a SIGHUP signal followed by a
 18962 SIGCONT signal shall be sent to each process in the newly-orphaned process group.
- 18963 • All open named semaphores in the calling process shall be closed as if by appropriate calls
 18964 to *sem_close()*.
- 18965 ML • Any memory locks established by the process via calls to *mlockall()* or *mlock()* shall be
 18966 removed. If locked pages in the address space of the calling process are also mapped into
 18967 the address spaces of other processes and are locked by those processes, the locks
 18968 established by the other processes shall be unaffected by the call by this process to *_Exit()*
 18969 or *_exit()*.
- 18970 • Memory mappings that were created in the process shall be unmapped before the process
 18971 is destroyed.
- 18972 TYM • Any blocks of typed memory that were mapped in the calling process shall be unmapped,
 18973 as if *munmap()* was implicitly called to unmap them.
- 18974 MSG • All open message queue descriptors in the calling process shall be closed as if by
 18975 appropriate calls to *mq_close()*.
- 18976 • Any outstanding cancelable asynchronous I/O operations may be canceled. Those
 18977 asynchronous I/O operations that are not canceled shall complete as if the *_Exit()* or
 18978 *_exit()* operation had not yet occurred, but any associated signal notifications shall be
 18979 suppressed. The *_Exit()* or *_exit()* operation may block awaiting such I/O completion.
 18980 Whether any I/O is canceled, and which I/O may be canceled upon *_Exit()* or *_exit()*, is
 18981 implementation-defined.
- 18982 • Threads terminated by a call to *_Exit()* or *_exit()* shall not invoke their cancellation
 18983 cleanup handlers or per-thread data destructors.
- 18984 OB TRC • If the calling process is a trace controller process, any trace streams that were created by
 18985 the calling process shall be shut down as described by the *posix_trace_shutdown()* function,
 18986 and mapping of trace event names to trace event type identifiers of any process built for
 18987 these trace streams may be deallocated.

RETURN VALUE

These functions do not return.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Normally applications should use *exit()* rather than *_Exit()* or *_exit()*.

RATIONALE**Process Termination**

Early proposals drew a distinction between normal and abnormal process termination. Abnormal termination was caused only by certain signals and resulted in implementation-defined “actions”, as discussed below. Subsequent proposals distinguished three types of termination: *normal termination* (as in the current specification), *simple abnormal termination*, and *abnormal termination with actions*. Again the distinction between the two types of abnormal termination was that they were caused by different signals and that implementation-defined actions would result in the latter case. Given that these actions were completely implementation-defined, the early proposals were only saying when the actions could occur and how their occurrence could be detected, but not what they were. This was of little or no use to conforming applications, and thus the distinction is not made in this volume of POSIX.1-2008.

The implementation-defined actions usually include, in most historical implementations, the creation of a file named **core** in the current working directory of the process. This file contains an image of the memory of the process, together with descriptive information about the process, perhaps sufficient to reconstruct the state of the process at the receipt of the signal.

There is a potential security problem in creating a **core** file if the process was set-user-ID and the current user is not the owner of the program, if the process was set-group-ID and none of the user’s groups match the group of the program, or if the user does not have permission to write in the current directory. In this situation, an implementation either should not create a **core** file or should make it unreadable by the user.

Despite the silence of this volume of POSIX.1-2008 on this feature, applications are advised not to create files named **core** because of potential conflicts in many implementations. Some implementations use a name other than **core** for the file; for example, by appending the process ID to the filename.

Terminating a Process

It is important that the consequences of process termination as described occur regardless of whether the process called *_exit()* (perhaps indirectly through *exit()*) or instead was terminated due to a signal or for some other reason. Note that in the specific case of *exit()* this means that the *status* argument to *exit()* is treated in the same way as the *status* argument to *_exit()*.

A language other than C may have other termination primitives than the C-language *exit()* function, and programs written in such a language should use its native termination primitives, but those should have as part of their function the behavior of *_exit()* as described. Implementations in languages other than C are outside the scope of this version of this volume of POSIX.1-2008, however.

As required by the ISO C standard, using **return** from *main()* has the same behavior (other than

_Exit()

with respect to language scope issues) as calling *exit()* with the returned value. Reaching the end of the *main()* function has the same behavior as calling *exit(0)*.

A value of zero (or *EXIT_SUCCESS*, which is required to be zero) for the argument *status* conventionally indicates successful termination. This corresponds to the specification for *exit()* in the ISO C standard. The convention is followed by utilities such as *make* and various shells, which interpret a zero status from a child process as success. For this reason, applications should not call *exit(0)* or *_exit(0)* when they terminate unsuccessfully; for example, in signal-catching functions.

Historically, the implementation-defined process that inherits children whose parents have terminated without waiting on them is called *init* and has a process ID of 1.

The sending of a *SIGHUP* to the foreground process group when a controlling process terminates corresponds to somewhat different historical implementations. In System V, the kernel sends a *SIGHUP* on termination of (essentially) a controlling process. In 4.2 BSD, the kernel does not send *SIGHUP* in a case like this, but the termination of a controlling process is usually noticed by a system daemon, which arranges to send a *SIGHUP* to the foreground process group with the *vhangup()* function. However, in 4.2 BSD, due to the behavior of the shells that support job control, the controlling process is usually a shell with no other processes in its process group. Thus, a change to make *_exit()* behave this way in such systems should not cause problems with existing applications.

The termination of a process may cause a process group to become orphaned in either of two ways. The connection of a process group to its parent(s) outside of the group depends on both the parents and their children. Thus, a process group may be orphaned by the termination of the last connecting parent process outside of the group or by the termination of the last direct descendant of the parent process(es). In either case, if the termination of a process causes a process group to become orphaned, processes within the group are disconnected from their job control shell, which no longer has any information on the existence of the process group. Stopped processes within the group would languish forever. In order to avoid this problem, newly orphaned process groups that contain stopped processes are sent a *SIGHUP* signal and a *SIGCONT* signal to indicate that they have been disconnected from their session. The *SIGHUP* signal causes the process group members to terminate unless they are catching or ignoring *SIGHUP*. Under most circumstances, all of the members of the process group are stopped if any of them are stopped.

The action of sending a *SIGHUP* and a *SIGCONT* signal to members of a newly orphaned process group is similar to the action of 4.2 BSD, which sends *SIGHUP* and *SIGCONT* to each stopped child of an exiting process. If such children exit in response to the *SIGHUP*, any additional descendants receive similar treatment at that time. In this volume of POSIX.1-2008, the signals are sent to the entire process group at the same time. Also, in this volume of POSIX.1-2008, but not in 4.2 BSD, stopped processes may be orphaned, but may be members of a process group that is not orphaned; therefore, the action taken at *_exit()* must consider processes other than child processes.

It is possible for a process group to be orphaned by a call to *setpgid()* or *setsid()*, as well as by process termination. This volume of POSIX.1-2008 does not require sending *SIGHUP* and *SIGCONT* in those cases, because, unlike process termination, those cases are not caused accidentally by applications that are unaware of job control. An implementation can choose to send *SIGHUP* and *SIGCONT* in those cases as an extension; such an extension must be documented as required in **<signal.h>**.

The ISO/IEC 9899:1999 standard adds the *_Exit()* function that results in immediate program termination without triggering signals or *atexit()*-registered functions. In POSIX.1-2008, this is

19080 equivalent to the `_exit()` function.

19081 FUTURE DIRECTIONS

19082 None.

19083 SEE ALSO

19084 `atexit()`, `exit()`, `mlock()`, `mlockall()`, `mq_close()`, `munmap()`, `posix_trace_create()`, `sem_close()`,
19085 `semop()`, `setpgid()`, `setsid()`, `shmget()`, `wait()`, `waitid()`

19086 XBD `<stdlib.h>`, `<unistd.h>`

19087 CHANGE HISTORY

19088 First released in Issue 1. Derived from Issue 1 of the SVID.

19089 Issue 5

19090 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
19091 Threads Extension.

19092 Interactions with the SA_NOCLDWAIT flag and SIGCHLD signal are further clarified.

19093 The values of *status* from `exit()` are better described.

19094 Issue 6

19095 Extensions beyond the ISO C standard are marked.

19096 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics
19097 for typed memory.

19098 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 19099 • The `_Exit()` function is included.
- 19100 • The DESCRIPTION is updated.

19101 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

19102 References to the `wait3()` function are removed.

19103 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/16 is applied, correcting grammar in the
19104 DESCRIPTION.

19105 Issue 7

19106 Austin Group Interpretation 1003.1-2001 #031 is applied, separating these functions from the
19107 `exit()` function.

19108 Austin Group Interpretation 1003.1-2001 #085 is applied, clarifying the text regarding flushing of
19109 streams and closing of temporary files.

19110 Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, and
19111 Semaphores options is moved to the Base.

_longjmp()

System Interfaces

19112 **NAME**

19113 _longjmp, _setjmp — non-local goto

19114 **SYNOPSIS**

```

19115 OB XSI #include <setjmp.h>
19116 void _longjmp(jmp_buf env, int val);
19117 int _setjmp(jmp_buf env);

```

19118 **DESCRIPTION**

19119 The *_longjmp()* and *_setjmp()* functions shall be equivalent to *longjmp()* and *setjmp()*,
 19120 respectively, with the additional restriction that *_longjmp()* and *_setjmp()* shall not manipulate
 19121 the signal mask.

19122 If *_longjmp()* is called even though *env* was never initialized by a call to *_setjmp()*, or when the
 19123 last such call was in a function that has since returned, the results are undefined.

19124 **RETURN VALUE**19125 Refer to *longjmp()* and *setjmp()*.19126 **ERRORS**

19127 No errors are defined.

19128 **EXAMPLES**

19129 None.

19130 **APPLICATION USAGE**

19131 If *_longjmp()* is executed and the environment in which *_setjmp()* was executed no longer exists,
 19132 errors can occur. The conditions under which the environment of the *_setjmp()* no longer exists
 19133 include exiting the function that contains the *_setjmp()* call, and exiting an inner block with
 19134 temporary storage. This condition might not be detectable, in which case the *_longjmp()* occurs
 19135 and, if the environment no longer exists, the contents of the temporary storage of an inner block
 19136 are unpredictable. This condition might also cause unexpected process termination. If the
 19137 function has returned, the results are undefined.

19138 Passing *longjmp()* a pointer to a buffer not created by *setjmp()*, passing *_longjmp()* a pointer to a
 19139 buffer not created by *_setjmp()*, passing *siglongjmp()* a pointer to a buffer not created by
 19140 *sigsetjmp()*, or passing any of these three functions a buffer that has been modified by the user
 19141 can cause all the problems listed above, and more.

19142 The *_longjmp()* and *_setjmp()* functions are included to support programs written to historical
 19143 system interfaces. New applications should use *siglongjmp()* and *sigsetjmp()* respectively.

19144 **RATIONALE**

19145 None.

19146 **FUTURE DIRECTIONS**19147 The *_longjmp()* and *_setjmp()* functions may be removed in a future version.19148 **SEE ALSO**19149 *longjmp()*, *setjmp()*, *siglongjmp()*, *sigsetjmp()*

19150 XBD <setjmp.h>

19151 **CHANGE HISTORY**

19152 First released in Issue 4, Version 2.

- 19153 **Issue 5**
19154 Moved from X/OPEN UNIX extension to BASE.
- 19155 **Issue 7**
19156 The *_longjmp()* and *_setjmp()* functions are marked obsolescent.

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

_tolower()*System Interfaces*19157 **NAME**

19158 _toupper — transliterate uppercase characters to lowercase

19159 **SYNOPSIS**

```
19160 OB XSI #include <ctype.h>
19161         int _tolower(int c);
```

19162 **DESCRIPTION**

19163 The *_tolower()* macro shall be equivalent to *tolower(c)* except that the application shall ensure
 19164 that the argument *c* is an uppercase letter.

19165 **RETURN VALUE**

19166 Upon successful completion, *_tolower()* shall return the lowercase letter corresponding to the
 19167 argument passed.

19168 **ERRORS**

19169 No errors are defined.

19170 **EXAMPLES**

19171 None.

19172 **APPLICATION USAGE**19173 Applications should use the *tolower()* function instead of the obsolescent *_tolower()* function.19174 **RATIONALE**

19175 None.

19176 **FUTURE DIRECTIONS**19177 The *_tolower()* function may be removed in a future version.19178 **SEE ALSO**19179 *tolower()*, *isupper()*19180 XBD Chapter 7 (on page 135), *<ctype.h>*19181 **CHANGE HISTORY**

19182 First released in Issue 1. Derived from Issue 1 of the SVID.

19183 **Issue 6**

19184 The normative text is updated to avoid use of the term “must” for application requirements.

19185 **Issue 7**19186 The *_tolower()* function is marked obsolescent.

19187 NAME

19188 _toupper — transliterate lowercase characters to uppercase

19189 SYNOPSIS

```
19190 OB XSI #include <ctype.h>
19191         int _toupper(int c);
```

19192 DESCRIPTION

19193 The *_toupper()* macro shall be equivalent to *toupper()* except that the application shall ensure
19194 that the argument *c* is a lowercase letter.

19195 RETURN VALUE

19196 Upon successful completion, *_toupper()* shall return the uppercase letter corresponding to the
19197 argument passed.

19198 ERRORS

19199 No errors are defined.

19200 EXAMPLES

19201 None.

19202 APPLICATION USAGE

19203 Applications should use the *toupper()* function instead of the obsolescent *_toupper()* function.

19204 RATIONALE

19205 None.

19206 FUTURE DIRECTIONS

19207 The *_toupper()* function may be removed in a future version.

19208 SEE ALSO

19209 *islower()*, *toupper()*

19210 XBD Chapter 7 (on page 135), *<ctype.h>*

19211 CHANGE HISTORY

19212 First released in Issue 1. Derived from Issue 1 of the SVID.

19213 Issue 6

19214 The normative text is updated to avoid use of the term “must” for application requirements.

19215 Issue 7

19216 The *_toupper()* function is marked obsolescent.

a64l()*System Interfaces*19217 **NAME**

19218 a64l, l64a — convert between a 32-bit integer and a radix-64 ASCII string

19219 **SYNOPSIS**

```
19220 XSI      #include <stdlib.h>
19221          long a64l(const char *s);
19222          char *l64a(long value);
```

19223 **DESCRIPTION**

19224 These functions maintain numbers stored in radix-64 ASCII characters. This is a notation by
 19225 which 32-bit integers can be represented by up to six characters; each character represents a digit
 19226 in radix-64 notation. If the type **long** contains more than 32 bits, only the low-order 32 bits shall
 19227 be used for these operations.

19228 The characters used to represent digits are ' .' (dot) for 0, ' / ' for 1, ' 0 ' through ' 9 ' for [2,11],
 19229 ' A ' through ' Z ' for [12,37], and ' a ' through ' z ' for [38,63].

19230 The *a64l()* function shall take a pointer to a radix-64 representation, in which the first digit is the
 19231 least significant, and return the corresponding **long** value. If the string pointed to by *s* contains
 19232 more than six characters, *a64l()* shall use the first six. If the first six characters of the string
 19233 contain a null terminator, *a64l()* shall use only characters preceding the null terminator. The
 19234 *a64l()* function shall scan the character string from left to right with the least significant digit on
 19235 the left, decoding each character as a 6-bit radix-64 number. If the type **long** contains more than
 19236 32 bits, the resulting value is sign-extended. The behavior of *a64l()* is unspecified if *s* is a null
 19237 pointer or the string pointed to by *s* was not generated by a previous call to *l64a()*.

19238 The *l64a()* function shall take a **long** argument and return a pointer to the corresponding
 19239 radix-64 representation. The behavior of *l64a()* is unspecified if *value* is negative.

19240 The value returned by *l64a()* may be a pointer into a static buffer. Subsequent calls to *l64a()* may
 19241 overwrite the buffer.

19242 The *l64a()* function need not be thread-safe.

19243 **RETURN VALUE**

19244 Upon successful completion, *a64l()* shall return the **long** value resulting from conversion of the
 19245 input string. If a string pointed to by *s* is an empty string, *a64l()* shall return 0L.

19246 The *l64a()* function shall return a pointer to the radix-64 representation. If *value* is 0L, *l64a()* shall
 19247 return a pointer to an empty string.

19248 **ERRORS**

19249 No errors are defined.

19250 **EXAMPLES**

19251 None.

19252 **APPLICATION USAGE**

19253 If the type **long** contains more than 32 bits, the result of *a64l(l64a(x))* is *x* in the low-order 32 bits.

19254 **RATIONALE**

19255 This is not the same encoding as used by either encoding variant of the *uuencode* utility.

19256 **FUTURE DIRECTIONS**

19257 None.

19258 **SEE ALSO**19259 [strtoul\(\)](#)19260 XBD [<stdlib.h>](#)19261 XCU [uuencode](#)19262 **CHANGE HISTORY**

19263 First released in Issue 4, Version 2.

19264 **Issue 5**

19265 Moved from X/OPEN UNIX extension to BASE.

19266 Normative text previously in the APPLICATION USAGE section is moved to the
19267 DESCRIPTION.19268 A note indicating that the *l64a()* function need not be reentrant is added to the DESCRIPTION.19269 **Issue 7**

19270 Austin Group Interpretation 1003.1-2001 #156 is applied.

abort()*System Interfaces*19271 **NAME**

19272 abort — generate an abnormal process abort

19273 **SYNOPSIS**

19274 #include <stdlib.h>

19275 void abort(void);

19276 **DESCRIPTION**

19277 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 19278 conflict between the requirements described here and the ISO C standard is unintentional. This
 19279 volume of POSIX.1-2008 defers to the ISO C standard.

19280 The *abort()* function shall cause abnormal process termination to occur, unless the signal
 19281 SIGABRT is being caught and the signal handler does not return.

19282 CX The abnormal termination processing shall include the default actions defined for SIGABRT and
 19283 may include an attempt to effect *fclose()* on all open streams.

19284 The SIGABRT signal shall be sent to the calling process as if by means of *raise()* with the
 19285 argument SIGABRT.

19286 CX The status made available to *wait()*, *waitid()*, or *waitpid()* by *abort()* shall be that of a process
 19287 terminated by the SIGABRT signal. The *abort()* function shall override blocking or ignoring the
 19288 SIGABRT signal.

19289 **RETURN VALUE**19290 The *abort()* function shall not return.19291 **ERRORS**

19292 No errors are defined.

19293 **EXAMPLES**

19294 None.

19295 **APPLICATION USAGE**

19296 Catching the signal is intended to provide the application developer with a portable means to
 19297 abort processing, free from possible interference from any implementation-supplied functions.

19298 **RATIONALE**

19299 The ISO/IEC 9899:1999 standard requires the *abort()* function to be async-signal-safe. Since
 19300 POSIX.1-2008 defers to the ISO C standard, this required a change to the DESCRIPTION from
 19301 “shall include the effect of *fclose()*” to “may include an attempt to effect *fclose()*.”

19302 The revised wording permits some backwards-compatibility and avoids a potential deadlock
 19303 situation.

19304 The Open Group Base Resolution bwg2002-003 is applied, removing the following XSI shaded
 19305 paragraph from the DESCRIPTION:

19306 “On XSI-conformant systems, in addition the abnormal termination processing shall include the
 19307 effect of *fclose()* on message catalog descriptors.”

19308 There were several reasons to remove this paragraph:

- 19309 • No special processing of open message catalogs needs to be performed prior to abnormal
 19310 process termination.
- 19311 • The main reason to specifically mention that *abort()* includes the effect of *fclose()* on open
 19312 streams is to flush output queued on the stream. Message catalogs in this context are read-
 19313 only and, therefore, do not need to be flushed.

- 19314 • The effect of *fclose()* on a message catalog descriptor is unspecified. Message catalog
 19315 descriptors are allowed, but not required to be implemented using a file descriptor, but
 19316 there is no mention in POSIX.1-2008 of a message catalog descriptor using a standard I/O
 19317 stream FILE object as would be expected by *fclose()*.

19318 **FUTURE DIRECTIONS**

19319 None.

19320 **SEE ALSO**

19321 *exit()*, *kill()*, *raise()*, *signal()*, *wait()*, *waitid()*

19322 XBD **<stdlib.h>**

19323 **CHANGE HISTORY**

19324 First released in Issue 1. Derived from Issue 1 of the SVID.

19325 **Issue 6**

19326 Extensions beyond the ISO C standard are marked.

19327 Changes are made to the DESCRIPTION for alignment with the ISO/IEC 9899:1999 standard.

19328 The Open Group Base Resolution bwg2002-003 is applied.

19329 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/10 is applied, changing the
 19330 DESCRIPTION of abnormal termination processing and adding to the RATIONALE section.

19331 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/9 is applied, changing “implementation-
 19332 defined functions” to “implementation-supplied functions” in the APPLICATION USAGE
 19333 section.

abs()*System Interfaces*19334 **NAME**

19335 abs — return an integer absolute value

19336 **SYNOPSIS**

19337 #include <stdlib.h>

19338 int abs(int i);

19339 **DESCRIPTION**

19340 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 19341 conflict between the requirements described here and the ISO C standard is unintentional. This
 19342 volume of POSIX.1-2008 defers to the ISO C standard.

19343 The *abs()* function shall compute the absolute value of its integer operand, *i*. If the result cannot
 19344 be represented, the behavior is undefined.

19345 **RETURN VALUE**19346 The *abs()* function shall return the absolute value of its integer operand.19347 **ERRORS**

19348 No errors are defined.

19349 **EXAMPLES**

19350 None.

19351 **APPLICATION USAGE**

19352 In two's-complement representation, the absolute value of the negative integer with largest
 19353 magnitude {INT_MIN} might not be representable.

19354 **RATIONALE**

19355 None.

19356 **FUTURE DIRECTIONS**

19357 None.

19358 **SEE ALSO**19359 *fabs()*, *labs()*

19360 XBD <stdlib.h>

19361 **CHANGE HISTORY**

19362 First released in Issue 1. Derived from Issue 1 of the SVID.

19363 **Issue 6**

19364 Extensions beyond the ISO C standard are marked.

System Interfaces

accept()**NAME**

accept — accept a new connection on a socket

SYNOPSIS

```
#include <sys/socket.h>

int accept(int socket, struct sockaddr *restrict address,
           socklen_t *restrict address_len);
```

DESCRIPTION

The *accept()* function shall extract the first connection on the queue of pending connections, create a new socket with the same socket type protocol and address family as the specified socket, and allocate a new file descriptor for that socket.

The *accept()* function takes the following arguments:

<i>socket</i>	Specifies a socket that was created with <i>socket()</i> , has been bound to an address with <i>bind()</i> , and has issued a successful call to <i>listen()</i> .
<i>address</i>	Either a null pointer, or a pointer to a sockaddr structure where the address of the connecting socket shall be returned.
<i>address_len</i>	Points to a socklen_t structure which on input specifies the length of the supplied sockaddr structure, and on output specifies the length of the stored address.

If *address* is not a null pointer, the address of the peer for the accepted connection shall be stored in the **sockaddr** structure pointed to by *address*, and the length of this address shall be stored in the object pointed to by *address_len*.

If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated.

If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by *address* is unspecified.

If the listen queue is empty of connection requests and O_NONBLOCK is not set on the file descriptor for the socket, *accept()* shall block until a connection is present. If the *listen()* queue is empty of connection requests and O_NONBLOCK is set on the file descriptor for the socket, *accept()* shall fail and set *errno* to [EAGAIN] or [EWOULDBLOCK].

The accepted socket cannot itself accept more connections. The original socket remains open and can accept more connections.

RETURN VALUE

Upon successful completion, *accept()* shall return the non-negative file descriptor of the accepted socket. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *accept()* function shall fail if:

[EAGAIN] or [EWOULDBLOCK]

O_NONBLOCK is set for the socket file descriptor and no connections are present to be accepted.

[EBADF] The *socket* argument is not a valid file descriptor.

[ECONNABORTED]

A connection has been aborted.

accept()*System Interfaces*

19407	[EINTR]	The <i>accept()</i> function was interrupted by a signal that was caught before a valid connection arrived.
19408		
19409	[EINVAL]	The <i>socket</i> is not accepting connections.
19410	[EMFILE]	All file descriptors available to the process are currently open.
19411	[ENFILE]	The maximum number of file descriptors in the system are already open.
19412	[ENOBUFS]	No buffer space is available.
19413	[ENOMEM]	There was insufficient memory available to complete the operation.
19414	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
19415	[EOPNOTSUPP]	The <i>socket</i> type of the specified socket does not support accepting connections.
19416		
19417		The <i>accept()</i> function may fail if:
19418	OB XSR [EPROTO]	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.
19419		

EXAMPLES

19420 None.

APPLICATION USAGE

19423 When a connection is available, *select()* indicates that the file descriptor for the socket is ready
19424 for reading.

RATIONALE

19426 None.

FUTURE DIRECTIONS

19428 None.

SEE ALSO

19430 *bind()*, *connect()*, *listen()*, *socket()*

19431 XBD <sys/socket.h>

CHANGE HISTORY

19433 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19434 The **restrict** keyword is added to the *accept()* prototype for alignment with the
19435 ISO/IEC 9899:1999 standard.

Issue 7

19437 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

19438 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOBUFS]
19439 and [ENOMEM] errors to become “shall fail” errors.

19440 Functionality relating to XSI STREAMS is marked obsolescent.

NAME

`access`, `faccessat` — determine accessibility of a file relative to directory file descriptor

SYNOPSIS

```
#include <unistd.h>

int access(const char *path, int amode);
int faccessat(int fd, const char *path, int amode, int flag);
```

DESCRIPTION

The `access()` function shall check the file named by the pathname pointed to by the `path` argument for accessibility according to the bit pattern contained in `amode`, using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID.

The value of `amode` is either the bitwise-inclusive OR of the access permissions to be checked (R_OK, W_OK, X_OK) or the existence test (F_OK).

If any access permissions are checked, each shall be checked individually, as described in XBD Section 4.4 (on page 108), except that where that description refers to execute permission for a process with appropriate privileges, an implementation may indicate success for X_OK even if execute permission is not granted to any user.

The `faccessat()` function shall be equivalent to the `access()` function, except in the case where `path` specifies a relative path. In this case the file whose accessibility is to be determined shall be located relative to the directory associated with the file descriptor `fd` instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function shall not perform the check.

If `faccessat()` is passed the special value AT_FDCWD in the `fd` parameter, the current working directory is used and the behavior shall be identical to a call to `access()`.

Values for `flag` are constructed by a bitwise-inclusive OR of flags from the following list, defined in `<fcntl.h>`:

AT_EACCESS	The checks for accessibility are performed using the effective user and group IDs instead of the real user and group ID as required in a call to <code>access()</code> .

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return -1 and set `errno` to indicate the error.

ERRORS

These functions shall fail if:

[EACCES]	Permission bits of the file mode do not permit the requested access, or search permission is denied on a component of the path prefix.
[ELOOP]	A loop exists in symbolic links encountered during resolution of the <code>path</code> argument.
[ENAMETOOLONG]	The length of a component of a pathname is longer than {NAME_MAX}.
[ENOENT]	A component of <code>path</code> does not name an existing file or <code>path</code> is an empty string.
[ENOTDIR]	A component of the path prefix is not a directory, or the <code>path</code> argument contains at least one non- <code><slash></code> character and ends with one or more trailing <code><slash></code> characters and the last pathname component names an existing file

access()*System Interfaces*

19485		that is neither a directory nor a symbolic link to a directory.
19486	[EROFS]	Write access is requested for a file on a read-only file system.
19487		The <i>faccessat()</i> function shall fail if:
19488	[EACCES]	<i>fd</i> was not opened with O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
19489		
19490	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
19491		
19492		These functions may fail if:
19493	[EINVAL]	The value of the <i>amode</i> argument is invalid.
19494	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
19495		
19496	[ENAMETOOLONG]	
19497		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
19498		
19499		
19500	[ETXTBSY]	Write access is requested for a pure procedure (shared text) file that is being executed.
19501		
19502		The <i>faccessat()</i> function may fail if:
19503	[EINVAL]	The value of the <i>flag</i> argument is not valid.
19504	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is neither AT_FDCWD nor a file descriptor associated with a directory.
19505		

EXAMPLES**Testing for the Existence of a File**

The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```
#include <unistd.h>
...
int result;
const char *filename = "/tmp/myfile";

result = access (filename, F_OK);
```

APPLICATION USAGE

Additional values of *amode* other than the set defined in the description may be valid; for example, if a system has extended access controls.

The use of the AT_EACCESS value for *flag* enables functionality not available in *access()*.

RATIONALE

In early proposals, some inadequacies in the *access()* function led to the creation of an *eaccess()* function because:

1. Historical implementations of *access()* do not test file access correctly when the process' real user ID is superuser. In particular, they always return zero when testing execute permissions without regard to whether the file is executable.

2. The superuser has complete access to all files on a system. As a consequence, programs started by the superuser and switched to the effective user ID with lesser privileges cannot use *access()* to test their file access permissions.

However, the historical model of *eaccess()* does not resolve problem (1), so this volume of POSIX.1-2008 now allows *access()* to behave in the desired way because several implementations have corrected the problem. It was also argued that problem (2) is more easily solved by using *open()*, *chdir()*, or one of the *exec* functions as appropriate and responding to the error, rather than creating a new function that would not be as reliable. Therefore, *eaccess()* is not included in this volume of POSIX.1-2008.

The sentence concerning appropriate privileges and execute permission bits reflects the two possibilities implemented by historical implementations when checking superuser access for *X_OK*.

New implementations are discouraged from returning *X_OK* unless at least one execution permission bit is set.

The purpose of the *faccessat()* function is to enable the checking of the accessibility of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *access()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *faccessat()* function it can be guaranteed that the file tested for accessibility is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod(), *fstatat()*

XBD Section 4.4 (on page 108), *<fcntl.h>*, *<unistd.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretations 1003.1-2001 #046 and #143 are applied.

The *faccessat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made to allow a directory to be opened for searching.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

acos()*System Interfaces*19565 **NAME**

19566 acos, acosf, acosl — arc cosine functions

19567 **SYNOPSIS**

19568 #include <math.h>

19569 double acos(double x);

19570 float acosf(float x);

19571 long double acosl(long double x);

19572 **DESCRIPTION**

19573 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 19574 conflict between the requirements described here and the ISO C standard is unintentional. This
 19575 volume of POSIX.1-2008 defers to the ISO C standard.

19576 These functions shall compute the principal value of the arc cosine of their argument x . The
 19577 value of x should be in the range $[-1,1]$.

19578 An application wishing to check for error situations should set *errno* to zero and call
 19579 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 19580 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 19581 zero, an error has occurred.

19582 **RETURN VALUE**

19583 Upon successful completion, these functions shall return the arc cosine of x , in the range $[0,\pi]$
 19584 radians.

19585 MX For finite values of x not in the range $[-1,1]$, a domain error shall occur, and either a NaN (if
 19586 supported), or an implementation-defined value shall be returned.

19587 MX If x is NaN, a NaN shall be returned.

19588 If x is $+1$, $+0$ shall be returned.

19589 If x is $\pm\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 19590 defined value shall be returned.

19591 **ERRORS**

19592 These functions shall fail if:

19593 MX Domain Error The x argument is finite and is not in the range $[-1,1]$, or is $\pm\text{Inf}$.

19594 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 19595 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 19596 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 19597 shall be raised.

19598 **EXAMPLES**

19599 None.

19600 **APPLICATION USAGE**

19601 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 19602 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

19603 **RATIONALE**

19604 None.

FUTURE DIRECTIONS

None.

SEE ALSO

cos(), *feclearexcept()*, *fetestexcept()*, *isnan()*

XBD [Section 4.19](#) (on page 116), [<math.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *acosf()* and *acosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

acosh()*System Interfaces*19621 **NAME**

19622 acosh, acoshf, acoshl — inverse hyperbolic cosine functions

19623 **SYNOPSIS**

```
19624       #include <math.h>

19625       double acosh(double x);
19626       float acoshf(float x);
19627       long double acoshl(long double x);
```

19628 **DESCRIPTION**

19629 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 19630 conflict between the requirements described here and the ISO C standard is unintentional. This
 19631 volume of POSIX.1-2008 defers to the ISO C standard.

19632 These functions shall compute the inverse hyperbolic cosine of their argument x .

19633 An application wishing to check for error situations should set *errno* to zero and call
 19634 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 19635 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 19636 zero, an error has occurred.

19637 **RETURN VALUE**

19638 Upon successful completion, these functions shall return the inverse hyperbolic cosine of their
 19639 argument.

19640 **MX** For finite values of $x < 1$, a domain error shall occur, and either a NaN (if supported), or an
 19641 implementation-defined value shall be returned.

19642 **MX** If x is NaN, a NaN shall be returned.

19643 If x is +1, +0 shall be returned.

19644 If x is +Inf, +Inf shall be returned.

19645 If x is -Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-
 19646 defined value shall be returned.

19647 **ERRORS**

19648 These functions shall fail if:

19649 **MX** Domain Error The x argument is finite and less than +1.0, or is -Inf.

19650 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 19651 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* &
 19652 MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 19653 shall be raised.

19654 **EXAMPLES**

19655 None.

19656 **APPLICATION USAGE**

19657 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 19658 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

19659 **RATIONALE**

19660 None.

FUTURE DIRECTIONS

None.

SEE ALSO*cosh()*, *feclearexcept()*, *fetestexcept()*XBD [Section 4.19](#) (on page 116), [<math.h>](#)**CHANGE HISTORY**

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6The *acosh()* function is no longer marked as an extension.The *acoshf()* and *acoshl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

acosl()*System Interfaces*19678 **NAME**

19679 acosl — arc cosine functions

19680 **SYNOPSIS**

19681 #include <math.h>

19682 long double acosl(long double x);

19683 **DESCRIPTION**19684 Refer to *acos()*.

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

NAME

aio_cancel — cancel an asynchronous I/O request

SYNOPSIS

```
#include <aio.h>
```

```
int aio_cancel(int fildes, struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_cancel()* function shall attempt to cancel one or more asynchronous I/O requests currently outstanding against file descriptor *fildes*. The *aiocbp* argument points to the asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then all outstanding cancelable asynchronous I/O requests against *fildes* shall be canceled.

Normal asynchronous notification shall occur for asynchronous I/O operations that are successfully canceled. If there are requests that cannot be canceled, then the normal asynchronous completion process shall take place for those requests when they are completed.

For requested operations that are successfully canceled, the associated error status shall be set to [ECANCELED] and the return status shall be -1. For requested operations that are not successfully canceled, the *aiocbp* shall not be modified by *aio_cancel()*.

If *aiocbp* is not NULL, then if *fildes* does not have the same value as the file descriptor with which the asynchronous operation was initiated, unspecified results occur.

Which operations are cancelable is implementation-defined.

RETURN VALUE

The *aio_cancel()* function shall return the value AIO_CANCELED if the requested operation(s) were canceled. The value AIO_NOTCANCELED shall be returned if at least one of the requested operation(s) cannot be canceled because it is in progress. In this case, the state of the other operations, if any, referenced in the call to *aio_cancel()* is not indicated by the return value of *aio_cancel()*. The application may determine the state of affairs for these operations by using *aio_error()*. The value AIO_ALLDONE is returned if all of the operations have already completed. Otherwise, the function shall return -1 and set *errno* to indicate the error.

ERRORS

The *aio_cancel()* function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_read(), *aio_write()*

XBD *<aio.h>*

aio_cancel()*System Interfaces***CHANGE HISTORY**

19726 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

19728 The [ENOSYS] error condition has been removed as stubs need not be provided if an
19729 implementation does not support the Asynchronous Input and Output option.
19730

19731 The APPLICATION USAGE section is added.

19732 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/10 is applied, removing the words “to the
19733 calling process” in the RETURN VALUE section. The term was unnecessary and precluded
19734 threads.

Issue 7

19735 The *aio_cancel()* function is moved from the Asynchronous Input and Output option to the Base.
19736

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

NAME

aio_error — retrieve errors status for an asynchronous I/O operation

SYNOPSIS

```
#include <aio.h>
```

```
int aio_error(const struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_error()* function shall return the error status associated with the **aiocb** structure referenced by the *aiocbp* argument. The error status for an asynchronous I/O operation is the *errno* value that would be set by the corresponding *read()*, *write()*, *fdatasync()*, or *fsync()* operation. If the operation has not yet completed, then the error status shall be equal to [EINPROGRESS].

If the **aiocb** structure pointed to by *aiocbp* is not associated with an operation that has been scheduled, the results are undefined.

RETURN VALUE

If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the asynchronous operation has completed unsuccessfully, then the error status, as described for *read()*, *write()*, *fdatasync()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has not yet completed, then [EINPROGRESS] shall be returned.

If the *aio_error()* function fails, it shall return -1 and set *errno* to indicate the error.

ERRORS

The *aio_error()* function may fail if:

[EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose return status has not yet been retrieved.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_cancel(), *aio_fsync()*, *aio_read()*, *aio_return()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*, *lseek()*, *read()*

XBD <aio.h>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The APPLICATION USAGE section is added.

aio_error()*System Interfaces***Issue 7**

- 19778 Austin Group Interpretation 1003.1-2001 #045 is applied.
- 19779
- 19780 SD5-XSH-ERN-148 is applied.
- 19781 The *aio_error()* function is moved from the Asynchronous Input and Output option to the Base.

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

NAME

aio_fsync — asynchronous file synchronization

SYNOPSIS

#include <aio.h>

int aio_fsync(int op, struct aiocb *aiocbp);

DESCRIPTION

The *aio_fsync()* function shall asynchronously force all I/O operations associated with the file indicated by the file descriptor *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync()* to the synchronized I/O completion state. The function call shall return when the synchronization request has been initiated or queued to the file or device (even when the data cannot be synchronized immediately).

If *op* is O_DSYNC, all currently queued I/O operations shall be completed as if by a call to *fdatasync()*; that is, as defined for synchronized I/O data integrity completion. If *op* is O_SYNC, all currently queued I/O operations shall be completed as if by a call to *fsync()*; that is, as defined for synchronized I/O file integrity completion. If the *aio_fsync()* function fails, or if the operation queued by *aio_fsync()* fails, then, as for *fsync()* and *fdatasync()*, outstanding I/O operations are not guaranteed to have been completed.

If *aio_fsync()* succeeds, then it is only the I/O that was queued at the time of the call to *aio_fsync()* that is guaranteed to be forced to the relevant completion state. The completion of subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized fashion.

The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used as an argument to *aio_error()* and *aio_return()* in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding. When the request is queued, the error status for the operation is [EINPROGRESS]. When all data has been successfully transferred, the error status shall be reset to reflect the success or failure of the operation. If the operation does not complete successfully, the error status for the operation shall be set to indicate the error. The *aio_sigevent* member determines the asynchronous notification to occur as specified in [Section 2.4.1](#) (on page 484) when all operations have achieved synchronized I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.

If the *aio_fsync()* function fails or *aiocbp* indicates an error condition, data is not guaranteed to have been successfully transferred.

RETURN VALUE

The *aio_fsync()* function shall return the value 0 if the I/O operation is successfully queued; otherwise, the function shall return the value -1 and set *errno* to indicate the error.

ERRORS

The *aio_fsync()* function shall fail if:

[EAGAIN]	The requested asynchronous operation was not queued due to temporary resource limitations.
[EBADF]	The <i>aio_fildes</i> member of the aiocb structure referenced by the <i>aiocbp</i> argument is not a valid file descriptor open for writing.
[EINVAL]	This implementation does not support synchronized I/O for this file.

aio_fsync()*System Interfaces*

19826 [EINVAL] A value of *op* other than O_DSYNC or O_SYNC was specified.

19827 In the event that any of the queued I/O operations fail, *aio_fsync()* shall return the error

19828 condition defined for *read()* and *write()*. The error is returned in the error status for the

19829 asynchronous *fsync()* operation, which can be retrieved using *aio_error()*.

19830 **EXAMPLES**

19831 None.

19832 **APPLICATION USAGE**

19833 None.

19834 **RATIONALE**

19835 None.

19836 **FUTURE DIRECTIONS**

19837 None.

19838 **SEE ALSO**

19839 *fcntl()*, *fdatasync()*, *fsync()*, *open()*, *read()*, *write()*

19840 XBD <*aio.h*>

19841 **CHANGE HISTORY**

19842 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

19843 **Issue 6**

19844 The [ENOSYS] error condition has been removed as stubs need not be provided if an

19845 implementation does not support the Asynchronous Input and Output option.

19846 The APPLICATION USAGE section is added.

19847 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/11 is applied, removing the words “to the

19848 calling process” in the RETURN VALUE section. The term was unnecessary and precluded

19849 threads.

19850 **Issue 7**

19851 The *aio_fsync()* function is moved from the Asynchronous Input and Output option to the Base.

System Interfaces

aioread()**NAME**

aioread — asynchronous read from a file

SYNOPSIS

```
#include <aiio.h>

int aio_read(struct aiocb *aiocbp);
```

DESCRIPTION

The *aioread()* function shall read *aiocbp->aio_nbytes* from the file associated with *aiocbp->aio_fildes* into the buffer pointed to by *aiocbp->aio_buf*. The function call shall return when the read request has been initiated or queued to the file or device (even when the data cannot be delivered immediately).

PIO

If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted at a priority equal to a base scheduling priority minus *aiocbp->aio_reqprio*. If Thread Execution Scheduling is not supported, then the base scheduling priority is that of the calling process;

PIO TPS

otherwise, the base scheduling priority is that of the calling thread.

The *aiocbp* value may be used as an argument to *aio_error()* and *aio_return()* in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding. If an error condition is encountered during queuing, the function call shall return without having initiated or queued the request. The requested operation takes place at the absolute position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to *SEEK_SET*. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified.

The *aio_sigevent* member specifies the notification which occurs when the request is completed.

The *aiocbp->aio_lio_opcode* field shall be ignored by *aioread()*.

The *aiocbp* argument points to an *aiocb* structure. If the buffer pointed to by *aiocbp->aio_buf* or the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.

Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

SIO

If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this function shall be according to the definitions of synchronized I/O data integrity completion and synchronized I/O file integrity completion.

For any system action that changes the process memory space while an asynchronous I/O is outstanding to the address range being changed, the result of that action is undefined.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *aiocbp->aio_fildes*.

RETURN VALUE

The *aioread()* function shall return the value zero if the I/O operation is successfully queued; otherwise, the function shall return the value -1 and set *errno* to indicate the error.

ERRORS

The *aioread()* function shall fail if:

[EAGAIN] The requested asynchronous I/O operation was not queued due to system resource limitations.

Each of the following conditions may be detected synchronously at the time of the call to *aioread()*, or asynchronously. If any of the conditions below are detected synchronously, the

aio_read()*System Interfaces*

- 19896 *aio_read()* function shall return `-1` and set *errno* to the corresponding value. If any of the
 19897 conditions below are detected asynchronously, the return status of the asynchronous operation
 19898 is set to `-1`, and the error status of the asynchronous operation is set to the corresponding value.
- 19899 [EBADF] The *aioctx*→*aio_fildes* argument is not a valid file descriptor open for reading.
- 19900 [EINVAL] The file offset value implied by *aioctx*→*aio_offset* would be invalid,
 19901 PIO *aioctx*→*aio_reqprio* is not a valid value, or *aioctx*→*aio_nbytes* is an invalid
 19902 value.
- 19903 In the case that the *aio_read()* successfully queues the I/O operation but the operation is
 19904 subsequently canceled or encounters an error, the return status of the asynchronous operation is
 19905 one of the values normally returned by the *read()* function call. In addition, the error status of
 19906 the asynchronous operation is set to one of the error statuses normally set by the *read()* function
 19907 call, or one of the following values:
- 19908 [EBADF] The *aioctx*→*aio_fildes* argument is not a valid file descriptor open for reading.
- 19909 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
 19910 *aio_cancel()* request.
- 19911 [EINVAL] The file offset value implied by *aioctx*→*aio_offset* would be invalid.
- 19912 The following condition may be detected synchronously or asynchronously:
- 19913 [EOVERFLOW] The file is a regular file, *aioctx*→*aio_nbytes* is greater than 0, and the starting
 19914 offset in *aioctx*→*aio_offset* is before the end-of-file and is at or beyond the
 19915 offset maximum in the open file description associated with *aioctx*→*aio_fildes*.

EXAMPLES

19916 None.

APPLICATION USAGE

19919 None.

RATIONALE

19921 None.

FUTURE DIRECTIONS

19923 None.

SEE ALSO

19925 *aio_cancel()*, *aio_error()*, *lio_listio()*, *aio_return()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,
 19926 *read()*

19927 XBD <*aio.h*>

CHANGE HISTORY

19929 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

19930 Large File Summit extensions are added.

Issue 6

19932 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 19933 implementation does not support the Asynchronous Input and Output option.

19934 The APPLICATION USAGE section is added.

- 19935 The following new requirements on POSIX implementations derive from alignment with the
19936 Single UNIX Specification:
- 19937 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open
19938 file description. This change is to support large files.
 - 19939 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support
19940 large files.
- 19941 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/12 is applied, rewording the
19942 DESCRIPTION when prioritized I/O is supported to account for threads, and removing the
19943 words “to the calling process” in the RETURN VALUE section.
- 19944 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/13 is applied, updating the [EINVAL]
19945 error, so that detection of an [EINVAL] error for an invalid value of *aiochp*→*aio_reqprio* is only
19946 required if the Prioritized Input and Output option is supported.
- 19947 **Issue 7**
- 19948 Austin Group Interpretation 1003.1-2001 #082 is applied.
- 19949 The *aio_read()* function is moved from the Asynchronous Input and Output option to the Base.

aio_return()

System Interfaces

19950 **NAME**

19951 aio_return — retrieve return status of an asynchronous I/O operation

19952 **SYNOPSIS**

19953 #include <aio.h>

19954 ssize_t aio_return(struct aiocb *aiocbp);

19955 **DESCRIPTION**

19956 The *aio_return()* function shall return the return status associated with the **aiocb** structure
 19957 referenced by the *aiocbp* argument. The return status for an asynchronous I/O operation is the
 19958 value that would be returned by the corresponding *read()*, *write()*, or *fsync()* function call. If the
 19959 error status for the operation is equal to [EINPROGRESS], then the return status for the
 19960 operation is undefined. The *aio_return()* function may be called exactly once to retrieve the
 19961 return status of a given asynchronous operation; thereafter, if the same **aiocb** structure is used in
 19962 a call to *aio_return()* or *aio_error()*, an error may be returned. When the **aiocb** structure referred
 19963 to by *aiocbp* is used to submit another asynchronous operation, then *aio_return()* may be
 19964 successfully used to retrieve the return status of that operation.

19965 **RETURN VALUE**

19966 If the asynchronous I/O operation has completed, then the return status, as described for *read()*,
 19967 *write()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has not yet completed,
 19968 the results of *aio_return()* are undefined.

19969 If the *aio_return()* function fails, it shall return -1 and set *errno* to indicate the error.

19970 **ERRORS**19971 The *aio_return()* function may fail if:

19972 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose
 19973 return status has not yet been retrieved.

19974 **EXAMPLES**

19975 None.

19976 **APPLICATION USAGE**

19977 None.

19978 **RATIONALE**

19979 None.

19980 **FUTURE DIRECTIONS**

19981 None.

19982 **SEE ALSO**

19983 *aio_cancel()*, *aio_error()*, *aio_fsync()*, *aio_read()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*,
 19984 *lseek()*, *read()*

19985 XBD <aio.h>

19986 **CHANGE HISTORY**

19987 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

19988 **Issue 6**

19989 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 19990 implementation does not support the Asynchronous Input and Output option.

19991 The APPLICATION USAGE section is added.

19992 The [EINVAL] error condition is made optional. This is for consistency with the DESCRIPTION.

19993 **Issue 7**

19994 SD5-XSH-ERN-148 is applied.

19995 The *aio_return()* function is moved from the Asynchronous Input and Output option to the Base.

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

aio_suspend()*System Interfaces*19996 **NAME**

19997 aio_suspend — wait for an asynchronous I/O request

19998 **SYNOPSIS**

19999 #include <aio.h>

```
20000 int aio_suspend(const struct aiocb *const list[], int nent,
20001               const struct timespec *timeout);
```

20002 **DESCRIPTION**

20003 The *aio_suspend()* function shall suspend the calling thread until at least one of the asynchronous
 20004 I/O operations referenced by the *list* argument has completed, until a signal interrupts the
 20005 function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any
 20006 of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is,
 20007 the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, the
 20008 function shall return without suspending the calling thread. The *list* argument is an array of
 20009 pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of
 20010 elements in the array. Each **aiocb** structure pointed to has been used in initiating an
 20011 asynchronous I/O request via *aio_read()*, *aio_write()*, or *lio_listio()*. This array may contain null
 20012 pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures that have
 20013 not been used in submitting asynchronous I/O, the effect is undefined.

20014 If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of
 20015 the I/O operations referenced by *list* are completed, then *aio_suspend()* shall return with an error.

20016 MON If the Monotonic Clock option is supported, the clock that shall be used to measure this time
 20017 interval shall be the CLOCK_MONOTONIC clock.

20018 **RETURN VALUE**

20019 If the *aio_suspend()* function returns after one or more asynchronous I/O operations have
 20020 completed, the function shall return zero. Otherwise, the function shall return a value of -1 and
 20021 set *errno* to indicate the error.

20022 The application may determine which asynchronous I/O completed by scanning the associated
 20023 error and return status using *aio_error()* and *aio_return()*, respectively.

20024 **ERRORS**

20025 The *aio_suspend()* function shall fail if:

20026 [EAGAIN] No asynchronous I/O indicated in the list referenced by *list* completed in the
 20027 time interval indicated by *timeout*.

20028 [EINTR] A signal interrupted the *aio_suspend()* function. Note that, since each
 20029 asynchronous I/O operation may possibly provoke a signal when it
 20030 completes, this error return may be caused by the completion of one (or more)
 20031 of the very I/O operations being awaited.

20032 **EXAMPLES**

20033 None.

20034 **APPLICATION USAGE**

20035 None.

20036 **RATIONALE**

20037 None.

20038 **FUTURE DIRECTIONS**

20039 None.

20040 **SEE ALSO**20041 *aio_read()*, *aio_write()*, *lio_listio()*

20042 XBD <aio.h>

20043 **CHANGE HISTORY**

20044 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

20045 **Issue 6**20046 The [ENOSYS] error condition has been removed as stubs need not be provided if an
20047 implementation does not support the Asynchronous Input and Output option.

20048 The APPLICATION USAGE section is added.

20049 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the
20050 CLOCK_MONOTONIC clock, if supported, is used.20051 **Issue 7**20052 The *aio_suspend()* function is moved from the Asynchronous Input and Output option to the
20053 Base.

aio_write()*System Interfaces*20054 **NAME**

20055 aio_write — asynchronous write to a file

20056 **SYNOPSIS**

20057 #include <aio.h>

20058 int aio_write(struct aiocb *aiocbp);

20059 **DESCRIPTION**

20060 The *aio_write()* function shall write *aiocbp->aio_nbytes* to the file associated with
 20061 *aiocbp->aio_fildes* from the buffer pointed to by *aiocbp->aio_buf*. The function shall return when
 20062 the write request has been initiated or, at a minimum, queued to the file or device.

20063 **PIO** If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted
 20064 at a priority equal to a base scheduling priority minus *aiocbp->aio_reqprio*. If Thread Execution
 20065 Scheduling is not supported, then the base scheduling priority is that of the calling process;
 20066 **PIO TPS** otherwise, the base scheduling priority is that of the calling thread.

20067 The *aiocbp* argument may be used as an argument to *aio_error()* and *aio_return()* in order to
 20068 determine the error status and return status, respectively, of the asynchronous operation while it
 20069 is proceeding.

20070 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio_buf* or
 20071 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
 20072 completion, then the behavior is undefined.

20073 If O_APPEND is not set for the file descriptor *aio_fildes*, then the requested operation shall take
 20074 place at the absolute position in the file as given by *aio_offset*, as if *lseek()* were called
 20075 immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to
 20076 SEEK_SET. If O_APPEND is set for the file descriptor, write operations append to the file in the
 20077 same order as the calls were made. After a successful call to enqueue an asynchronous I/O
 20078 operation, the value of the file offset for the file is unspecified.

20079 The *aio_sigevent* member specifies the notification which occurs when the request is completed.

20080 The *aiocbp->aio_lio_opcode* field shall be ignored by *aio_write()*.

20081 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

20082 **SIO** If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this
 20083 function shall be according to the definitions of synchronized I/O data integrity completion, and
 20084 synchronized I/O file integrity completion.

20085 For any system action that changes the process memory space while an asynchronous I/O is
 20086 outstanding to the address range being changed, the result of that action is undefined.

20087 For regular files, no data transfer shall occur past the offset maximum established in the open
 20088 file description associated with *aiocbp->aio_fildes*.

20089 **RETURN VALUE**

20090 The *aio_write()* function shall return the value zero if the I/O operation is successfully queued;
 20091 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

20092 **ERRORS**

20093 The *aio_write()* function shall fail if:

20094 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
 20095 resource limitations.

20096 Each of the following conditions may be detected synchronously at the time of the call to
 20097 *aio_write()*, or asynchronously. If any of the conditions below are detected synchronously, the

System Interfaces

aio_write()

20098 *aio_write()* function shall return `-1` and set *errno* to the corresponding value. If any of the
 20099 conditions below are detected asynchronously, the return status of the asynchronous operation
 20100 shall be set to `-1`, and the error status of the asynchronous operation is set to the corresponding
 20101 value.

20102 [EBADF] The *aiocbp*→*aio_fildes* argument is not a valid file descriptor open for writing.

20103 [EINVAL] The file offset value implied by *aiocbp*→*aio_offset* would be invalid,
 20104 PIO *aiocbp*→*aio_reqprio* is not a valid value, or *aiocbp*→*aio_nbytes* is an invalid
 20105 value.

20106 In the case that the *aio_write()* successfully queues the I/O operation, the return status of the
 20107 asynchronous operation shall be one of the values normally returned by the *write()* function call.
 20108 If the operation is successfully queued but is subsequently canceled or encounters an error, the
 20109 error status for the asynchronous operation contains one of the values normally set by the
 20110 *write()* function call, or one of the following:

20111 [EBADF] The *aiocbp*→*aio_fildes* argument is not a valid file descriptor open for writing.

20112 [EINVAL] The file offset value implied by *aiocbp*→*aio_offset* would be invalid.

20113 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
 20114 *aio_cancel()* request.

20115 The following condition may be detected synchronously or asynchronously:

20116 [EFBIG] The file is a regular file, *aiocbp*→*aio_nbytes* is greater than 0, and the starting
 20117 offset in *aiocbp*→*aio_offset* is at or beyond the offset maximum in the open file
 20118 description associated with *aiocbp*→*aio_fildes*.

20119 **EXAMPLES**

20120 None.

20121 **APPLICATION USAGE**

20122 None.

20123 **RATIONALE**

20124 None.

20125 **FUTURE DIRECTIONS**

20126 None.

20127 **SEE ALSO**

20128 *aio_cancel()*, *aio_error()*, *aio_read()*, *aio_return()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*, *lseek()*,
 20129 *write()*

20130 XBD <*aio.h*>

20131 **CHANGE HISTORY**

20132 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

20133 Large File Summit extensions are added.

20134 **Issue 6**

20135 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 20136 implementation does not support the Asynchronous Input and Output option.

20137 The APPLICATION USAGE section is added.

20138 The following new requirements on POSIX implementations derive from alignment with the
 20139 Single UNIX Specification:

aio_write()*System Interfaces*

- 20140 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs
 20141 past the offset maximum established in the open file description associated with
 20142 *aioctx*→*aio_fildes*.
- 20143 • The [EFBIG] error is added as part of the large file support extensions.
- 20144 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/14 is applied, rewording the
 20145 DESCRIPTION when prioritized I/O is supported to account for threads, and removing the
 20146 words “to the calling process” in the RETURN VALUE section.
- 20147 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/15 is applied, updating the [EINVAL]
 20148 error, so that detection of an [EINVAL] error for an invalid value of *aioctx*→*aio_reqprio* is only
 20149 required if the Prioritized Input and Output option is supported.
- 20150 **Issue 7**
- 20151 Austin Group Interpretation 1003.1-2001 #082 is applied.
- 20152 The *aio_write()* function is moved from the Asynchronous Input and Output option to the Base.

System Interfaces

alarm()**NAME**

alarm — schedule an alarm signal

SYNOPSIS

```
#include <unistd.h>

unsigned alarm(unsigned seconds);
```

DESCRIPTION

The *alarm()* function shall cause the system to generate a SIGALRM signal for the process after the number of realtime seconds specified by *seconds* have elapsed. Processor scheduling delays may prevent the process from handling the signal as soon as it is generated.

If *seconds* is 0, a pending alarm request, if any, is canceled.

Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner. If the SIGALRM signal has not yet been generated, the call shall result in rescheduling the time at which the SIGALRM signal is generated.

XSI Interactions between *alarm()* and *setitimer()* are unspecified.

RETURN VALUE

If there is a previous *alarm()* request with time remaining, *alarm()* shall return a non-zero value that is the number of seconds until the previous request would have generated a SIGALRM signal. Otherwise, *alarm()* shall return 0.

ERRORS

The *alarm()* function is always successful, and no return value is reserved to indicate an error.

EXAMPLES

None.

APPLICATION USAGE

The *fork()* function clears pending alarms in the child process. A new process image created by one of the *exec* functions inherits the time left to an alarm signal in the image of the old process.

Application developers should note that the type of the argument *seconds* and the return value of *alarm()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX}, which the ISO C standard sets as 65 535, and any application passing a larger value is restricting its portability. A different type was considered, but historical implementations, including those with a 16-bit **int** type, consistently use either **unsigned** or **int**.

Application developers should be aware of possible interactions when the same process uses both the *alarm()* and *sleep()* functions.

RATIONALE

Many historical implementations (including Version 7 and System V) allow an alarm to occur up to a second early. Other implementations allow alarms up to half a second or one clock tick early or do not allow them to occur early at all. The latter is considered most appropriate, since it gives the most predictable behavior, especially since the signal can always be delayed for an indefinite amount of time due to scheduling. Applications can thus choose the *seconds* argument as the minimum amount of time they wish to have elapse before the signal.

The term “realtime” here and elsewhere (*sleep()*, *times()*) is intended to mean “wall clock” time as common English usage, and has nothing to do with “realtime operating systems”. It is in contrast to *virtual time*, which could be misinterpreted if just *time* were used.

In some implementations, including 4.3 BSD, very large values of the *seconds* argument are silently rounded down to an implementation-specific maximum value. This maximum is large

alarm()*System Interfaces*

- 20198 enough (to the order of several months) that the effect is not noticeable.
- 20199 There were two possible choices for alarm generation in multi-threaded applications: generation
 20200 for the calling thread or generation for the process. The first option would not have been
 20201 particularly useful since the alarm state is maintained on a per-process basis and the alarm that
 20202 is established by the last invocation of *alarm()* is the only one that would be active.
- 20203 Furthermore, allowing generation of an asynchronous signal for a thread would have
 20204 introduced an exception to the overall signal model. This requires a compelling reason in order
 20205 to be justified.
- 20206 **FUTURE DIRECTIONS**
- 20207 None.
- 20208 **SEE ALSO**
- 20209 *alarm()*, *exec*, *fork()*, *getitimer()*, *pause()*, *sigaction()*, *sleep()*
- 20210 XBD *<signal.h>*, *<unistd.h>*
- 20211 **CHANGE HISTORY**
- 20212 First released in Issue 1. Derived from Issue 1 of the SVID.
- 20213 **Issue 6**
- 20214 The following new requirements on POSIX implementations derive from alignment with the
 20215 Single UNIX Specification:
- 20216 • The DESCRIPTION is updated to indicate that interactions with the *setitimer()*, *ualarm()*,
 20217 and *usleep()* functions are unspecified.
- 20218 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/16 is applied, replacing “an
 20219 implementation-defined maximum value” with “an implementation-specific maximum value”
 20220 in the RATIONALE.

NAME

alphasort, scandir — scan a directory

SYNOPSIS

```
#include <dirent.h>

int alphasort(const struct dirent **d1, const struct dirent **d2);
int scandir(const char *dir, struct dirent ***namelist,
            int (*sel)(const struct dirent *),
            int (*compar)(const struct dirent **, const struct dirent **));
```

DESCRIPTION

The *alphasort()* function can be used as the comparison function for the *scandir()* function to sort the directory entries, *d1* and *d2*, into alphabetical order. Sorting happens as if by calling the *strcoll()* function on the *d_name* element of the **dirent** structures passed as the two parameters. If the *strcoll()* function fails, the return value of *alphasort()* is unspecified.

The *alphasort()* function shall not change the setting of *errno* if successful. Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *alphasort()*, then check *errno*.

The *scandir()* function shall scan the directory *dir*, calling the function referenced by *sel* on each directory entry. Entries for which the function referenced by *sel* returns non-zero shall be stored in strings allocated as if by a call to *malloc()*, and sorted as if by a call to *qsort()* with the comparison function *compar*, except that *compar* need not provide total ordering. The strings are collected in array *namelist* which shall be allocated as if by a call to *malloc()*. If *sel* is a null pointer, all entries shall be selected. If the comparison function *compar* does not provide total ordering, the order in which the directory entries are stored is unspecified.

RETURN VALUE

Upon successful completion, the *alphasort()* function shall return an integer greater than, equal to, or less than 0, according to whether the name of the directory entry pointed to by *d1* is lexicographically greater than, equal to, or less than the directory pointed to by *d2* when both are interpreted as appropriate to the current locale. There is no return value reserved to indicate an error.

Upon successful completion, the *scandir()* function shall return the number of entries in the array and a pointer to the array through the parameter *namelist*. Otherwise, the *scandir()* function shall return -1.

ERRORS

The *scandir()* function shall fail if:

[EACCES]	Search permission is denied for the component of the path prefix of <i>dir</i> or read permission is denied for <i>dir</i> .
[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>dir</i> argument.
[ENAMETOOLONG]	The length of a component of a pathname is longer than {NAME_MAX}.
[ENOENT]	A component of <i>dir</i> does not name an existing directory or <i>dir</i> is an empty string.
[ENOMEM]	Insufficient storage space is available.

alphasort()*System Interfaces*

- 20264 [ENOTDIR] A component of *dir* is not a directory.
- 20265 The *scandir()* function may fail if:
- 20266 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
20267 resolution of the *dir* argument.
- 20268 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.
- 20269 [ENAMETOOLONG]
20270 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
20271 symbolic link produced an intermediate result with a length that exceeds
20272 {PATH_MAX}.
- 20273 [ENFILE] Too many files are currently open in the system.

EXAMPLES

20274 An example to print the files in the current directory:

```
20276 #include <dirent.h>
20277 #include <stdio.h>
20278 #include <stdlib.h>
20279 ...
20280 struct dirent **namelist;
20281 int i,n;

20282     n = scandir(".", &namelist, 0, alphasort);
20283     if (n < 0)
20284         perror("scandir");
20285     else {
20286         for (i = 0; i < n; i++) {
20287             printf("%s\n", namelist[i]->d_name);
20288             free(namelist[i]);
20289         }
20290     }
20291     free(namelist);
20292     ...
```

APPLICATION USAGE

20293 If *dir* contains filenames that contain characters outside the domain of the collating sequence of
20294 the current locale, the *alphasort()* function need not provide a total ordering.

20296 The *scandir()* function may allocate dynamic storage during its operation. If *scandir()* is forcibly
20297 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *sel*
20298 or *compar*, or by an interrupt routine, *scandir()* does not have a chance to free that storage, so it
20299 remains permanently allocated. A safe way to handle interrupts is to store the fact that an
20300 interrupt has occurred, then wait until *scandir()* returns to act on the interrupt.

20301 For functions that allocate memory as if by *malloc()*, the application should release such memory
20302 when it is no longer required by a call to *free()*. For *scandir()*, this is *namelist* (including all of the
20303 individual strings in *namelist*).

RATIONALE

20304 None.

20306 **FUTURE DIRECTIONS**

20307 None.

20308 **SEE ALSO**20309 *qsort()*, *strcoll()*20310 XBD **<dirent.h>**20311 **CHANGE HISTORY**

20312 First released in Issue 7.

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

asctime()*System Interfaces*20313 **NAME**20314 `asctime, asctime_r` — convert date and time to a string20315 **SYNOPSIS**

```

20316 OB      #include <time.h>
20317          char *asctime(const struct tm *timeptr);
20318 OB CX     char *asctime_r(const struct tm *restrict tm, char *restrict buf);

```

20319 **DESCRIPTION**

20320 CX For `asctime()`: The functionality described on this reference page is aligned with the ISO C
 20321 standard. Any conflict between the requirements described here and the ISO C standard is
 20322 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

20323 The `asctime()` function shall convert the broken-down time in the structure pointed to by `timeptr`
 20324 into a string in the form:

20325 Sun Sep 16 01:03:52 1973\n\0

20326 using the equivalent of the following algorithm:

```

20327 char *asctime(const struct tm *timeptr)
20328 {
20329     static char wday_name[7][3] = {
20330         "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
20331     };
20332     static char mon_name[12][3] = {
20333         "Jan", "Feb", "Mar", "Apr", "May", "Jun",
20334         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
20335     };
20336     static char result[26];
20337     sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
20338         wday_name[timeptr->tm_wday],
20339         mon_name[timeptr->tm_mon],
20340         timeptr->tm_mday, timeptr->tm_hour,
20341         timeptr->tm_min, timeptr->tm_sec,
20342         1900 + timeptr->tm_year);
20343     return result;
20344 }

```

20345 However, the behavior is undefined if `timeptr->tm_wday` or `timeptr->tm_mon` are not within the
 20346 normal ranges as defined in `<time.h>`, or if `timeptr->tm_year` exceeds `{INT_MAX}-1990`, or if the
 20347 above algorithm would attempt to generate more than 26 bytes of output (including the
 20348 terminating null).

20349 The `tm` structure is defined in the `<time.h>` header.

20350 CX The `asctime()`, `ctime()`, `gmtime()`, and `localtime()` functions shall return values in one of two static
 20351 objects: a broken-down time structure and an array of type `char`. Execution of any of the
 20352 functions may overwrite the information returned in either of these objects by any of the other
 20353 functions.

20354 The `asctime()` function need not be thread-safe.

20355 The `asctime_r()` function shall convert the broken-down time in the structure pointed to by `tm`
 20356 into a string (of the same form as that returned by `asctime()`, and with the same undefined
 20357 behavior when input or output is out of range) that is placed in the user-supplied buffer pointed

20358 to by *buf* (which shall contain at least 26 bytes) and then return *buf*.

20359 RETURN VALUE

20360 CX Upon successful completion, *asctime()* shall return a pointer to the string. If the function is
20361 unsuccessful, it shall return NULL.

20362 Upon successful completion, *asctime_r()* shall return a pointer to a character string containing
20363 the date and time. This string is pointed to by the argument *buf*. If the function is unsuccessful,
20364 it shall return NULL.

20365 ERRORS

20366 No errors are defined.

20367 EXAMPLES

20368 None.

20369 APPLICATION USAGE

20370 These functions are included only for compatibility with older implementations. They have
20371 undefined behavior if the resulting string would be too long, so the use of these functions
20372 should be discouraged. On implementations that do not detect output string length overflow, it
20373 is possible to overflow the output buffers in such a way as to cause applications to fail, or
20374 possible system security violations. Also, these functions do not support localized date and time
20375 formats. To avoid these problems, applications should use *strftime()* to generate strings from
20376 broken-down times.

20377 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.

20378 The *asctime_r()* function is thread-safe and shall return values in a user-supplied buffer instead
20379 of possibly using a static data area that may be overwritten by each call.

20380 RATIONALE

20381 The standard developers decided to mark the *asctime()* and *asctime_r()* functions obsolescent
20382 even though they are in the ISO C standard due to the possibility of buffer overflow. The ISO C
20383 standard also provides the *strftime()* function which can be used to avoid these problems.

20384 FUTURE DIRECTIONS

20385 These functions may be removed in a future version.

20386 SEE ALSO

20387 *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*

20388 XBD <time.h>

20389 CHANGE HISTORY

20390 First released in Issue 1. Derived from Issue 1 of the SVID.

20391 Issue 5

20392 Normative text previously in the APPLICATION USAGE section is moved to the
20393 DESCRIPTION.

20394 The *asctime_r()* function is included for alignment with the POSIX Threads Extension.

20395 A note indicating that the *asctime()* function need not be reentrant is added to the
20396 DESCRIPTION.

20397 Issue 6

20398 The *asctime_r()* function is marked as part of the Thread-Safe Functions option.

20399 Extensions beyond the ISO C standard are marked.

20400 The APPLICATION USAGE section is updated to include a note on the thread-safe function and

asctime()*System Interfaces*

- 20401 its avoidance of possibly using a static data area.
- 20402 The DESCRIPTION of *asctime_r()* is updated to describe the format of the string returned.
- 20403 The **restrict** keyword is added to the *asctime_r()* prototype for alignment with the
- 20404 ISO/IEC 9899:1999 standard
- 20405 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/17 is applied, adding the CX extension in
- 20406 the RETURN VALUE section requiring that if the *asctime()* function is unsuccessful it returns
- 20407 NULL.
- 20408 **Issue 7**
- 20409 Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent.
- 20410 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 20411 The *asctime_r()* function is moved from the Thread-Safe Functions option to the Base.

20412 **NAME**

20413 asin, asinf, asinl — arc sine function

20414 **SYNOPSIS**

```
20415 #include <math.h>
20416 double asin(double x);
20417 float asinf(float x);
20418 long double asinl(long double x);
```

20419 **DESCRIPTION**

20420 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20421 conflict between the requirements described here and the ISO C standard is unintentional. This
 20422 volume of POSIX.1-2008 defers to the ISO C standard.

20423 These functions shall compute the principal value of the arc sine of their argument x . The value
 20424 of x should be in the range $[-1,1]$.

20425 An application wishing to check for error situations should set *errno* to zero and call
 20426 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 20427 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 20428 zero, an error has occurred.

20429 **RETURN VALUE**

20430 Upon successful completion, these functions shall return the arc sine of x , in the range
 20431 $[-\pi/2, \pi/2]$ radians.

20432 MX For finite values of x not in the range $[-1,1]$, a domain error shall occur, and either a NaN (if
 20433 supported), or an implementation-defined value shall be returned.

20434 MX If x is NaN, a NaN shall be returned.

20435 If x is ± 0 , x shall be returned.

20436 If x is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 20437 defined value shall be returned.

20438 If x is subnormal, a range error may occur and x should be returned.

20439 **ERRORS**

20440 These functions shall fail if:

20441 MX Domain Error The x argument is finite and is not in the range $[-1,1]$, or is $\pm \text{Inf}$.

20442 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20443 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 20444 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 20445 shall be raised.

20446 These functions may fail if:

20447 MX Range Error The value of x is subnormal.

20448 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20449 then *errno* shall be set to [ERANGE]. If the integer expression
 20450 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 20451 floating-point exception shall be raised.

asin()*System Interfaces*20452 **EXAMPLES**

20453 None.

20454 **APPLICATION USAGE**

20455 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
20456 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

20457 **RATIONALE**

20458 None.

20459 **FUTURE DIRECTIONS**

20460 None.

20461 **SEE ALSO**20462 *feclearexcept()*, *fetestexcept()*, *isnan()*, *sin()*

20463 XBD Section 4.19 (on page 116), <math.h>

20464 **CHANGE HISTORY**

20465 First released in Issue 1. Derived from Issue 1 of the SVID.

20466 **Issue 5**

20467 The DESCRIPTION is updated to indicate how an application should check for an error. This
20468 text was previously published in the APPLICATION USAGE section.

20469 **Issue 6**20470 The *asinf()* and *asinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

20471 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
20472 revised to align with the ISO/IEC 9899:1999 standard.

20473 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
20474 marked.

NAME

asinh, asinhf, asinhl — inverse hyperbolic sine functions

SYNOPSIS

```
#include <math.h>

double asinh(double x);
float asinhf(float x);
long double asinhl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall compute the inverse hyperbolic sine of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the inverse hyperbolic sine of their argument.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ± 0 , or $\pm \text{Inf}$, *x* shall be returned.

If *x* is subnormal, a range error may occur and *x* should be returned.

ERRORS

These functions may fail if:

MX **Range Error** The value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *fetestexcept()*, *sinh()*

XBD Section 4.19 (on page 116), *<math.h>*

asinh()

System Interfaces

20516 **CHANGE HISTORY**

20517 First released in Issue 4, Version 2.

20518 **Issue 5**

20519 Moved from X/OPEN UNIX extension to BASE.

20520 **Issue 6**20521 The *asinh()* function is no longer marked as an extension.20522 The *asinhf()* and *asinhll()* functions are added for alignment with the ISO/IEC 9899:1999
20523 standard.20524 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
20525 revised to align with the ISO/IEC 9899:1999 standard.20526 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
20527 marked.

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

*System Interfaces***asinl()**20528 **NAME**20529 **asinl** — arc sine function20530 **SYNOPSIS**20531 `#include <math.h>`20532 `long double asinl(long double x);`20533 **DESCRIPTION**20534 Refer to *asin()*.

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

assert()*System Interfaces*20535 **NAME**20536 **assert** — insert program diagnostics20537 **SYNOPSIS**20538 `#include <assert.h>`20539 `void assert(scalar expression);`20540 **DESCRIPTION**

20541 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 20542 conflict between the requirements described here and the ISO C standard is unintentional. This
 20543 volume of POSIX.1-2008 defers to the ISO C standard.

20544 The `assert()` macro shall insert diagnostics into programs; it shall expand to a **void** expression.
 20545 When it is executed, if *expression* (which shall have a **scalar** type) is false (that is, compares equal
 20546 to 0), `assert()` shall write information about the particular call that failed on *stderr* and shall call
 20547 `abort()`.

20548 The information written about the call that failed shall include the text of the argument, the
 20549 name of the source file, the source file line number, and the name of the enclosing function; the
 20550 latter are, respectively, the values of the preprocessing macros `__FILE__` and `__LINE__` and of
 20551 the identifier `__func__`.

20552 Forcing a definition of the name `NDEBUG`, either from the compiler command line or with the
 20553 preprocessor control statement `#define NDEBUG` ahead of the `#include <assert.h>` statement,
 20554 shall stop assertions from being compiled into the program.

20555 **RETURN VALUE**20556 The `assert()` macro shall not return a value.20557 **ERRORS**

20558 No errors are defined.

20559 **EXAMPLES**

20560 None.

20561 **APPLICATION USAGE**

20562 None.

20563 **RATIONALE**

20564 None.

20565 **FUTURE DIRECTIONS**

20566 None.

20567 **SEE ALSO**20568 `abort()`, `stdin`20569 `XBD <assert.h>`20570 **CHANGE HISTORY**

20571 First released in Issue 1. Derived from Issue 1 of the SVID.

20572 **Issue 6**

20573 The prototype for the *expression* argument to `assert()` is changed from **int** to **scalar** for alignment
 20574 with the ISO/IEC 9899:1999 standard.

20575 The DESCRIPTION of `assert()` is updated for alignment with the ISO/IEC 9899:1999 standard.

NAME

atan, atanf, atanl — arc tangent function

SYNOPSIS

```
#include <math.h>

double atan(double x);
float atanf(float x);
long double atanl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall compute the principal value of the arc tangent of their argument x .

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc tangent of x in the range $[-\pi/2, \pi/2]$ radians.

MX If x is NaN, a NaN shall be returned.

If x is ± 0 , x shall be returned.

If x is $\pm \text{Inf}$, $\pm \pi/2$ shall be returned.

If x is subnormal, a range error may occur and x should be returned.

ERRORS

These functions may fail if:

MX **Range Error** The value of x is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atan2(), *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*

XBD Section 4.19 (on page 116), **<math.h>**

atan()*System Interfaces*20618 **CHANGE HISTORY**

20619 First released in Issue 1. Derived from Issue 1 of the SVID.

20620 **Issue 5**20621 The DESCRIPTION is updated to indicate how an application should check for an error. This
20622 text was previously published in the APPLICATION USAGE section.20623 **Issue 6**20624 The *atanf()* and *atanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.20625 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
20626 revised to align with the ISO/IEC 9899:1999 standard.20627 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
20628 marked.

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

NAME

atan2, atan2f, atan2l — arc tangent functions

SYNOPSIS

```
#include <math.h>

double atan2(double y, double x);
float atan2f(float y, float x);
long double atan2l(long double y, long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall compute the principal value of the arc tangent of y/x , using the signs of both arguments to determine the quadrant of the return value.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc tangent of y/x in the range $[-\pi, \pi]$ radians.

If y is ± 0 and x is < 0 , $\pm\pi$ shall be returned.

If y is ± 0 and x is > 0 , ± 0 shall be returned.

If y is < 0 and x is ± 0 , $-\pi/2$ shall be returned.

If y is > 0 and x is ± 0 , $\pi/2$ shall be returned.

If x is 0, a pole error shall not occur.

MX If either x or y is NaN, a NaN shall be returned.

If the result underflows, a range error may occur and y/x should be returned.

If y is ± 0 and x is -0 , $\pm\pi$ shall be returned.

If y is ± 0 and x is $+0$, ± 0 shall be returned.

For finite values of $\pm y > 0$, if x is $-\text{Inf}$, $\pm\pi$ shall be returned.

For finite values of $\pm y > 0$, if x is $+\text{Inf}$, ± 0 shall be returned.

For finite values of x , if y is $\pm\text{Inf}$, $\pm\pi/2$ shall be returned.

If y is $\pm\text{Inf}$ and x is $-\text{Inf}$, $\pm 3\pi/4$ shall be returned.

If y is $\pm\text{Inf}$ and x is $+\text{Inf}$, $\pm\pi/4$ shall be returned.

If both arguments are 0, a domain error shall not occur.

ERRORS

These functions may fail if:

MX **Range Error** The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression

atan2()*System Interfaces*

20669 *(math_errhandling & MATH_ERREXCEPT)* is non-zero, then the underflow
 20670 floating-point exception shall be raised.

20671 **EXAMPLES**20672 **Converting Cartesian to Polar Coordinates System**

20673 The function below uses *atan2()* to convert a 2d vector expressed in cartesian coordinates (*x,y*) to
 20674 the polar coordinates (*rho,theta*). There are other ways to compute the angle *theta*, using *asin()*
 20675 *acos()*, or *atan()*. However, *atan2()* presents here two advantages:

- 20676 • The angle's quadrant is automatically determined.
- 20677 • The singular cases (0,*y*) are taken into account.

20678 Finally, this example uses *hypot()* rather than *sqrt()* since it is better for special cases; see *hypot()*
 20679 for more information.

```
20680 #include <math.h>
20681 void
20682 cartesian_to_polar(const double x, const double y,
20683                  double *rho, double *theta)
20684 {
20685     {
20686         *rho = hypot (x,y); /* better than sqrt(x*x+y*y) */
20687         *theta = atan2 (y,x);
20688     }

```

20689 **APPLICATION USAGE**

20690 On error, the expressions *(math_errhandling & MATH_ERRNO)* and *(math_errhandling &*
 20691 *MATH_ERREXCEPT)* are independent of each other, but at least one of them must be non-zero.

20692 **RATIONALE**

20693 None.

20694 **FUTURE DIRECTIONS**

20695 None.

20696 **SEE ALSO**

20697 *acos()*, *asin()*, *atan()*, *feclearexcept()*, *fetestexcept()*, *hypot()*, *isnan()*, *sqrt()*, *tan()*

20698 XBD Section 4.19 (on page 116), **<math.h>**

20699 **CHANGE HISTORY**

20700 First released in Issue 1. Derived from Issue 1 of the SVID.

20701 **Issue 5**

20702 The DESCRIPTION is updated to indicate how an application should check for an error. This
 20703 text was previously published in the APPLICATION USAGE section.

20704 **Issue 6**

20705 The *atan2f()* and *atan2l()* functions are added for alignment with the ISO/IEC 9899:1999
 20706 standard.

20707 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 20708 revised to align with the ISO/IEC 9899:1999 standard, and the IEC 60559:1989 standard
 20709 floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

20710
20711

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/18 is applied, adding to the EXAMPLES section.

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

atanf()*System Interfaces*20712 **NAME**

20713 atanf — arc tangent function

20714 **SYNOPSIS**

20715 #include <math.h>

20716 float atanf(float x);

20717 **DESCRIPTION**20718 Refer to *atan()*.

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

System Interfaces

atanh()20719 **NAME**

20720 atanh, atanhf, atanhf — inverse hyperbolic tangent functions

20721 **SYNOPSIS**

```
20722       #include <math.h>

20723       double atanh(double x);
20724       float atanhf(float x);
20725       long double atanhf(long double x);
```

20726 **DESCRIPTION**

20727 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 20728 conflict between the requirements described here and the ISO C standard is unintentional. This
 20729 volume of POSIX.1-2008 defers to the ISO C standard.

20730 These functions shall compute the inverse hyperbolic tangent of their argument x .

20731 An application wishing to check for error situations should set *errno* to zero and call
 20732 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 20733 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 20734 zero, an error has occurred.

20735 **RETURN VALUE**

20736 Upon successful completion, these functions shall return the inverse hyperbolic tangent of their
 20737 argument.

20738 If x is ± 1 , a pole error shall occur, and *atanh()*, *atanhf()*, and *atanf()* shall return the value of the
 20739 macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively, with the same sign as the
 20740 correct value of the function.

20741 **MX** For finite $|x| > 1$, a domain error shall occur, and either a NaN (if supported), or an
 20742 implementation-defined value shall be returned.

20743 **MX** If x is NaN, a NaN shall be returned.

20744 If x is ± 0 , x shall be returned.

20745 If x is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 20746 defined value shall be returned.

20747 If x is subnormal, a range error may occur and x should be returned.

20748 **ERRORS**

20749 These functions shall fail if:

20750 **MX** Domain Error The x argument is finite and not in the range $[-1, 1]$, or is $\pm \text{Inf}$.

20751 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20752 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 20753 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 20754 shall be raised.

20755 Pole Error The x argument is ± 1 .

20756 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20757 then *errno* shall be set to [ERANGE]. If the integer expression
 20758 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 20759 floating-point exception shall be raised.

atanh()*System Interfaces*

20760 These functions may fail if:

20761	MX	Range Error	The value of x is subnormal.
20762			If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
20763			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
20764			(<i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the underflow
20765			floating-point exception shall be raised.

20766 **EXAMPLES**

20767 None.

20768 **APPLICATION USAGE**

20769 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
20770 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

20771 **RATIONALE**

20772 None.

20773 **FUTURE DIRECTIONS**

20774 None.

20775 **SEE ALSO**

20776 *feclearexcept()*, *fetestexcept()*, *tanh()*

20777 XBD Section 4.19 (on page 116), **<math.h>**

20778 **CHANGE HISTORY**

20779 First released in Issue 4, Version 2.

20780 **Issue 5**

20781 Moved from X/OPEN UNIX extension to BASE.

20782 **Issue 6**

20783 The *atanh()* function is no longer marked as an extension.

20784 The *atanhf()* and *atanhl()* functions are added for alignment with the ISO/IEC 9899:1999
20785 standard.

20786 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
20787 revised to align with the ISO/IEC 9899:1999 standard.

20788 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
20789 marked.

*System Interfaces***atanl()**20790 **NAME**

20791 atanl — arc tangent function

20792 **SYNOPSIS**

20793 #include <math.h>

20794 long double atanl(long double x);

20795 **DESCRIPTION**20796 Refer to *atan()*.

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

atexit()*System Interfaces*20797 **NAME**20798 **atexit** — register a function to run at process termination20799 **SYNOPSIS**20800 `#include <stdlib.h>`20801 `int atexit(void (*func)(void));`20802 **DESCRIPTION**

20803 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20804 conflict between the requirements described here and the ISO C standard is unintentional. This
 20805 volume of POSIX.1-2008 defers to the ISO C standard.

20806 The *atexit()* function shall register the function pointed to by *func*, to be called without
 20807 arguments at normal program termination. At normal program termination, all functions
 20808 registered by the *atexit()* function shall be called, in the reverse order of their registration, except
 20809 that a function is called after any previously registered functions that had already been called at
 20810 the time it was registered. Normal termination occurs either by a call to *exit()* or a return from
 20811 *main()*.

20812 At least 32 functions can be registered with *atexit()*.

20813 CX After a successful call to any of the *exec* functions, any functions previously registered by *atexit()*
 20814 shall no longer be registered.

20815 **RETURN VALUE**

20816 Upon successful completion, *atexit()* shall return 0; otherwise, it shall return a non-zero value.

20817 **ERRORS**

20818 No errors are defined.

20819 **EXAMPLES**

20820 None.

20821 **APPLICATION USAGE**

20822 The functions registered by a call to *atexit()* must return to ensure that all registered functions
 20823 are called.

20824 The application should call *sysconf()* to obtain the value of {ATEXIT_MAX}, the number of
 20825 functions that can be registered. There is no way for an application to tell how many functions
 20826 have already been registered with *atexit()*.

20827 Since the behavior is undefined if the *exit()* function is called more than once, portable
 20828 applications calling *atexit()* must ensure that the *exit()* function is not called at normal process
 20829 termination when all functions registered by the *atexit()* function are called.

20830 All functions registered by the *atexit()* function are called at normal process termination, which
 20831 occurs by a call to the *exit()* function or a return from *main()* or on the last thread termination,
 20832 when the behavior is as if the implementation called *exit()* with a zero argument at thread
 20833 termination time.

20834 If, at normal process termination, a function registered by the *atexit()* function is called and a
 20835 portable application needs to stop further *exit()* processing, it must call the *_exit()* function or
 20836 the *_Exit()* function or one of the functions which cause abnormal process termination.

20837 **RATIONALE**

20838 None.

20839 **FUTURE DIRECTIONS**

20840 None.

20841 **SEE ALSO**20842 *exec*, *exit()*, *sysconf()*20843 XBD *<stdlib.h>*20844 **CHANGE HISTORY**

20845 First released in Issue 4. Derived from the ANSI C standard.

20846 **Issue 6**

20847 Extensions beyond the ISO C standard are marked.

20848 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

20849 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/19 is applied, adding further clarification
20850 to the APPLICATION USAGE section.

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

atof()*System Interfaces*20851 **NAME**20852 **atof** — convert a string to a double-precision number20853 **SYNOPSIS**20854 `#include <stdlib.h>`20855 `double atof(const char *str);`20856 **DESCRIPTION**

20857 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 20858 conflict between the requirements described here and the ISO C standard is unintentional. This
 20859 volume of POSIX.1-2008 defers to the ISO C standard.

20860 The call *atof(str)* shall be equivalent to:20861 `strtod(str, (char **)NULL),`

20862 except that the handling of errors may differ. If the value cannot be represented, the behavior is
 20863 undefined.

20864 **RETURN VALUE**20865 The *atof()* function shall return the converted value if the value can be represented.20866 **ERRORS**

20867 No errors are defined.

20868 **EXAMPLES**

20869 None.

20870 **APPLICATION USAGE**

20871 The *atof()* function is subsumed by *strtod()* but is retained because it is used extensively in
 20872 existing code. If the number is not known to be in range, *strtod()* should be used because *atof()*
 20873 is not required to perform any error checking.

20874 **RATIONALE**

20875 None.

20876 **FUTURE DIRECTIONS**

20877 None.

20878 **SEE ALSO**20879 [*strtod\(\)*](#)20880 XBD [*<stdlib.h>*](#)20881 **CHANGE HISTORY**

20882 First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces

atoi()**NAME**

atoi — convert a string to an integer

SYNOPSIS

```
#include <stdlib.h>

int atoi(const char *str);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The call *atoi(str)* shall be equivalent to:

```
(int) strtol(str, (char **)NULL, 10)
```

except that the handling of errors may differ. If the value cannot be represented, the behavior is undefined.

RETURN VALUE

The *atoi()* function shall return the converted value if the value can be represented.

ERRORS

No errors are defined.

EXAMPLES**Converting an Argument**

The following example checks for proper usage of the program. If there is an argument and the decimal conversion of this argument (obtained using *atoi()*) is greater than 0, then the program has a valid number of minutes to wait for an event.

```
#include <stdlib.h>
#include <stdio.h>
...
int minutes_to_event;
...
if (argc < 2 || ((minutes_to_event = atoi (argv[1]))) <= 0) {
    fprintf(stderr, "Usage: %s minutes\n", argv[0]); exit(1);
}
...
```

APPLICATION USAGE

The *atoi()* function is subsumed by *strtol()* but is retained because it is used extensively in existing code. If the number is not known to be in range, *strtol()* should be used because *atoi()* is not required to perform any error checking.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtol()

XBD *<stdlib.h>*

atoi()

System Interfaces

CHANGE HISTORY

20925
20926

First released in Issue 1. Derived from Issue 1 of the SVID.

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

20927 **NAME**20928 `atol`, `atoll` — convert a string to a long integer20929 **SYNOPSIS**20930 `#include <stdlib.h>`20931 `long atol(const char *str);`20932 `long long atoll(const char *nptr);`20933 **DESCRIPTION**

20934 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 20935 conflict between the requirements described here and the ISO C standard is unintentional. This
 20936 volume of POSIX.1-2008 defers to the ISO C standard.

20937 The call `atol(str)` shall be equivalent to:20938 `strtol(str, (char **)NULL, 10)`20939 The call `atoll(nptr)` shall be equivalent to:20940 `strtoll(nptr, (char **)NULL, 10)`

20941 except that the handling of errors may differ. If the value cannot be represented, the behavior is
 20942 undefined.

20943 **RETURN VALUE**

20944 These functions shall return the converted value if the value can be represented.

20945 **ERRORS**

20946 No errors are defined.

20947 **EXAMPLES**

20948 None.

20949 **APPLICATION USAGE**

20950 The `atol()` function is subsumed by `strtol()` but is retained because it is used extensively in
 20951 existing code. If the number is not known to be in range, `strtol()` should be used because `atol()` is
 20952 not required to perform any error checking.

20953 **RATIONALE**

20954 None.

20955 **FUTURE DIRECTIONS**

20956 None.

20957 **SEE ALSO**20958 `strtol()`20959 XBD `<stdlib.h>`20960 **CHANGE HISTORY**

20961 First released in Issue 1. Derived from Issue 1 of the SVID.

20962 **Issue 6**20963 The `atoll()` function is added for alignment with the ISO/IEC 9899:1999 standard.20964 **Issue 7**20965 SD5-XSH-ERN-61 is applied, correcting the DESCRIPTION of `atoll()`.

basename()

20966 **NAME**
20967 **basename** — return the last component of a pathname

20968 **SYNOPSIS**
20969 XSI

```
#include <libgen.h>
```


20970

```
char *basename(char *path);
```

20971 **DESCRIPTION**
20972 The *basename()* function shall take the pathname pointed to by *path* and return a pointer to the
20973 final component of the pathname, deleting any trailing '/' characters.

20974 If the string pointed to by *path* consists entirely of the '/' character, *basename()* shall return a
20975 pointer to the string "/". If the string pointed to by *path* is exactly "///", it is implementation-
20976 defined whether '/' or "///" is returned.

20977 If *path* is a null pointer or points to an empty string, *basename()* shall return a pointer to the
20978 string ".".

20979 The *basename()* function may modify the string pointed to by *path*, and may return a pointer to
20980 static storage that may then be overwritten by a subsequent call to *basename()*.

20981 The *basename()* function need not be thread-safe.

20982 **RETURN VALUE**
20983 The *basename()* function shall return a pointer to the final component of *path*.

20984 **ERRORS**
20985 No errors are defined.

20986 **EXAMPLES**

20987 **Using basename()**

20988 The following program fragment returns a pointer to the value *lib*, which is the base name of
20989 */usr/lib*.

20990

```
#include <libgen.h>
```


20991

```
...
```


20992

```
char *name = "/usr/lib";
```


20993

```
char *base;
```


20994

```
base = basename(name);
```


20995

```
...
```

20996 **Sample Input and Output Strings for basename()**
20997 In the following table, the input string is the value pointed to by *path*, and the output string is
20998 the return value of the *basename()* function.

Input String	Output String
"/usr/lib"	"lib"
"/usr/"	"usr"
"/"	"/"
"////"	"/"
"/usr//lib//"	"lib"

21005 APPLICATION USAGE

21006 None.

21007 RATIONALE

21008 None.

21009 FUTURE DIRECTIONS

21010 None.

21011 SEE ALSO21012 *dirname()*

21013 XBD <libgen.h>

21014 XCU *basename***21015 CHANGE HISTORY**

21016 First released in Issue 4, Version 2.

21017 Issue 5

21018 Moved from X/OPEN UNIX extension to BASE.

21019 Normative text previously in the APPLICATION USAGE section is moved to the
21020 DESCRIPTION.

21021 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

21022 Issue 6

21023 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

21024 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/20 is applied, changing the
21025 DESCRIPTION to make it clear that the string referenced is the string pointed to by *path*.**21026 Issue 7**

21027 Austin Group Interpretation 1003.1-2001 #156 is applied.

bind()*System Interfaces*21028 **NAME**21029 **bind** — bind a name to a socket21030 **SYNOPSIS**21031 `#include <sys/socket.h>`

21032 `int bind(int socket, const struct sockaddr *address,`
 21033 `socklen_t address_len);`

21034 **DESCRIPTION**

21035 The *bind()* function shall assign a local socket address *address* to a socket identified by descriptor
 21036 *socket* that has no local socket address assigned. Sockets created with the *socket()* function are
 21037 initially unnamed; they are identified only by their address family.

21038 The *bind()* function takes the following arguments:

21039 *socket* Specifies the file descriptor of the socket to be bound.

21040 *address* Points to a **sockaddr** structure containing the address to be bound to the
 21041 socket. The length and format of the address depend on the address family of
 21042 the socket.

21043 *address_len* Specifies the length of the **sockaddr** structure pointed to by the *address*
 21044 argument.

21045 The socket specified by *socket* may require the process to have appropriate privileges to use the
 21046 *bind()* function.

21047 If the socket address cannot be assigned immediately and `O_NONBLOCK` is set for the file
 21048 descriptor for the socket, *bind()* shall fail and set *errno* to `[EINPROGRESS]`, but the assignment
 21049 request shall not be aborted, and the assignment shall be completed asynchronously. Subsequent
 21050 calls to *bind()* for the same socket, before the assignment is completed, shall fail and set *errno* to
 21051 `[EALREADY]`.

21052 When the assignment has been performed asynchronously, *pselect()*, *select()*, and *poll()* shall
 21053 indicate that the file descriptor for the socket is ready for reading and writing.

21054 **RETURN VALUE**

21055 Upon successful completion, *bind()* shall return 0; otherwise, `-1` shall be returned and *errno* set
 21056 to indicate the error.

21057 **ERRORS**

21058 The *bind()* function shall fail if:

21059 `[EADDRINUSE]` The specified address is already in use.

21060 `[EADDRNOTAVAIL]`

21061 The specified address is not available from the local machine.

21062 `[EAFNOSUPPORT]`

21063 The specified address is not a valid address for the address family of the
 21064 specified socket.

21065 `[EALREADY]` An assignment request is already in progress for the specified socket.

21066 `[EBADF]` The *socket* argument is not a valid file descriptor.

21067 `[EINPROGRESS]` `O_NONBLOCK` is set for the file descriptor for the socket and the assignment
 21068 cannot be immediately performed; the assignment shall be performed
 21069 asynchronously.

System Interfaces

bind()

21070	[EINVAL]	The socket is already bound to an address, and the protocol does not support binding to a new address; or the socket has been shut down.
21071		
21072	[ENOBUFS]	Insufficient resources were available to complete the call.
21073	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
21074	[EOPNOTSUPP]	The socket type of the specified socket does not support binding to an address.
21075	If the address family of the socket is AF_UNIX, then <i>bind()</i> shall fail if:	
21076	[EACCES]	A component of the path prefix denies search permission, or the requested name requires writing in a directory with a mode that denies write permission.
21077		
21078		
21079	[EDESTADDRREQ] or [EISDIR]	
21080		The <i>address</i> argument is a null pointer.
21081	[EIO]	An I/O error occurred.
21082	[ELOOP]	A loop exists in symbolic links encountered during resolution of the pathname in <i>address</i> .
21083		
21084	[ENAMETOOLONG]	
21085		The length of a component of a pathname is longer than {NAME_MAX}.
21086	[ENOENT]	A component of the pathname does not name an existing file or the pathname is an empty string.
21087		
21088	[ENOTDIR]	A component of the path prefix of the pathname in <i>address</i> is not a directory, or the pathname in <i>address</i> contains at least one non- <i><slash></i> character and ends with one or more trailing <i><slash></i> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
21089		
21090		
21091		
21092		
21093	[EROFS]	The name would reside on a read-only file system.
21094	The <i>bind()</i> function may fail if:	
21095	[EACCES]	The specified address is protected and the current user does not have permission to bind to it.
21096		
21097	[EINVAL]	The <i>address_len</i> argument is not a valid length for the address family.
21098	[EISCONN]	The socket is already connected.
21099	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in <i>address</i> .
21100		
21101	[ENAMETOOLONG]	
21102		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
21103		
21104		

bind()*System Interfaces***EXAMPLES**

The following code segment shows how to create a socket and bind it to a name in the AF_UNIX domain.

```
#define MY_SOCKET_PATH "/somepath"

int sfd;
struct sockaddr_un my_addr;

sfd = socket(AF_UNIX, SOCK_STREAM, 0);
if (sfd == -1)
    /* Handle error */;

memset(&my_addr, '\0', sizeof(struct sockaddr_un));
/* Clear structure */
my_addr.sun_family = AF_UNIX;
strncpy(my_addr.sun_path, MY_SOCKET_PATH, sizeof(my_addr.sun_path) - 1);

if (bind(sfd, (struct sockaddr *) &my_addr,
        sizeof(struct sockaddr_un)) == -1)
    /* Handle error */;
```

APPLICATION USAGE

An application program can retrieve the assigned socket name with the *getsockname()* function.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

connect(), *getsockname()*, *listen()*, *socket()*

XBD [<sys/socket.h>](#)

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOBUFFS] error to become a “shall fail” error.

Austin Group Interpretation 1003.1-2001 #143 is applied.

SD5-XSH-ERN-185 is applied.

An example is added.

21138 **NAME**21139 **bsearch** — binary search a sorted table21140 **SYNOPSIS**

```
21141 #include <stdlib.h>
21142 void *bsearch(const void *key, const void *base, size_t nel,
21143             size_t width, int (*compar)(const void *, const void *));
```

21144 **DESCRIPTION**

21145 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 21146 conflict between the requirements described here and the ISO C standard is unintentional. This
 21147 volume of POSIX.1-2008 defers to the ISO C standard.

21148 The *bsearch()* function shall search an array of *nel* objects, the initial element of which is pointed
 21149 to by *base*, for an element that matches the object pointed to by *key*. The size of each element in
 21150 the array is specified by *width*. If the *nel* argument has the value zero, the comparison function
 21151 pointed to by *compar* shall not be called and no match shall be found.

21152 The comparison function pointed to by *compar* shall be called with two arguments that point to
 21153 the *key* object and to an array element, in that order.

21154 The application shall ensure that the comparison function pointed to by *compar* does not alter the
 21155 contents of the array. The implementation may reorder elements of the array between calls to the
 21156 comparison function, but shall not alter the contents of any individual element.

21157 The implementation shall ensure that the first argument is always a pointer to the key.

21158 When the same objects (consisting of *width* bytes, irrespective of their current positions in the
 21159 array) are passed more than once to the comparison function, the results shall be consistent with
 21160 one another. That is, the same object shall always compare the same way with the key.

21161 The application shall ensure that the function returns an integer less than, equal to, or greater
 21162 than 0 if the *key* object is considered, respectively, to be less than, to match, or to be greater than
 21163 the array element. The application shall ensure that the array consists of all the elements that
 21164 compare less than, all the elements that compare equal to, and all the elements that compare
 21165 greater than the *key* object, in that order.

21166 **RETURN VALUE**

21167 The *bsearch()* function shall return a pointer to a matching member of the array, or a null pointer
 21168 if no match is found. If two or more members compare equal, which member is returned is
 21169 unspecified.

21170 **ERRORS**

21171 No errors are defined.

21172 **EXAMPLES**

21173 The example below searches a table containing pointers to nodes consisting of a string and its
 21174 length. The table is ordered alphabetically on the string in the node pointed to by each entry.

21175 The code fragment below reads in strings and either finds the corresponding node and prints
 21176 out the string and its length, or prints an error message.

```
21177 #include <stdio.h>
21178 #include <stdlib.h>
21179 #include <string.h>
21180 #define TABSIZE      1000
21181 struct node {                /* These are stored in the table. */
```

bsearch()*System Interfaces*

```

21182     char *string;
21183     int length;
21184 };
21185 struct node table[TABSIZE];    /* Table to be searched. */
21186     .
21187     .
21188     .
21189 {
21190     struct node *node_ptr, node;
21191     /* Routine to compare 2 nodes. */
21192     int node_compare(const void *, const void *);
21193     .
21194     .
21195     .
21196     while (scanf("%ms", &node.string) != EOF) {
21197         node_ptr = (struct node *)bsearch((void *)(&node),
21198             (void *)table, TABSIZE,
21199             sizeof(struct node), node_compare);
21200         if (node_ptr != NULL) {
21201             (void)printf("string = %20s, length = %d\n",
21202                 node_ptr->string, node_ptr->length);
21203         } else {
21204             (void)printf("not found: %s\n", node.string);
21205         }
21206         free(node.string);
21207     }
21208 }
21209 /*
21210     This routine compares two nodes based on an
21211     alphabetical ordering of the string field.
21212 */
21213 int
21214 node_compare(const void *node1, const void *node2)
21215 {
21216     return strcmp(((const struct node *)node1)->string,
21217         ((const struct node *)node2)->string);
21218 }

```

APPLICATION USAGE

The pointers to the key and the element at the base of the table should be of type pointer-to-element.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

In practice, the array is usually sorted according to the comparison function.

RATIONALE

The requirement that the second argument (hereafter referred to as *p*) to the comparison function is a pointer to an element of the array implies that for every call all of the following expressions are non-zero:

```

((char *)p - (char *(base) % width == 0
(char *)p >= (char *)base

```


21231 (char *)p < (char *)base + nel * width

21232 **FUTURE DIRECTIONS**

21233 None.

21234 **SEE ALSO**

21235 *hcreate()*, *lsearch()*, *qsort()*, *tdelete()*

21236 XBD <stdlib.h>

21237 **CHANGE HISTORY**

21238 First released in Issue 1. Derived from Issue 1 of the SVID.

21239 **Issue 6**

21240 The normative text is updated to avoid use of the term “must” for application requirements.

21241 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/11 is applied, adding to the
 21242 DESCRIPTION the last sentence of the first non-shaded paragraph, and the following three
 21243 paragraphs. The RATIONALE section is also updated. These changes are for alignment with the
 21244 ISO C standard.

21245 **Issue 7**

21246 The EXAMPLES section is revised.

btowc()

System Interfaces

21247 **NAME**21248 **btowc** — single byte to wide character conversion21249 **SYNOPSIS**21250 `#include <stdio.h>`21251 `#include <wchar.h>`21252 `wint_t btowc(int c);`21253 **DESCRIPTION**

21254 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 21255 conflict between the requirements described here and the ISO C standard is unintentional. This
 21256 volume of POSIX.1-2008 defers to the ISO C standard.

21257 The *btowc()* function shall determine whether *c* constitutes a valid (one-byte) character in the
 21258 initial shift state.

21259 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

21260 **RETURN VALUE**

21261 The *btowc()* function shall return WEOF if *c* has the value EOF or if **(unsigned char)** *c* does not
 21262 constitute a valid (one-byte) character in the initial shift state. Otherwise, it shall return the
 21263 wide-character representation of that character.

21264 **ERRORS**

21265 No errors are defined.

21266 **EXAMPLES**

21267 None.

21268 **APPLICATION USAGE**

21269 None.

21270 **RATIONALE**

21271 None.

21272 **FUTURE DIRECTIONS**

21273 None.

21274 **SEE ALSO**21275 *wctob()*21276 XBD *<stdio.h>*, *<wchar.h>*21277 **CHANGE HISTORY**

21278 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 21279 (E).

21280 NAME

21281 cabs, cabsf, cabsl — return a complex absolute value

21282 SYNOPSIS

```
21283       #include <complex.h>

21284       double cabs(double complex z);
21285       float cabsf(float complex z);
21286       long double cabsl(long double complex z);
```

21287 DESCRIPTION

21288 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21289 conflict between the requirements described here and the ISO C standard is unintentional. This
 21290 volume of POSIX.1-2008 defers to the ISO C standard.

21291 These functions shall compute the complex absolute value (also called norm, modulus, or
 21292 magnitude) of *z*.

21293 RETURN VALUE

21294 These functions shall return the complex absolute value.

21295 ERRORS

21296 No errors are defined.

21297 EXAMPLES

21298 None.

21299 APPLICATION USAGE

21300 None.

21301 RATIONALE

21302 None.

21303 FUTURE DIRECTIONS

21304 None.

21305 SEE ALSO

21306 XBD [<complex.h>](#)

21307 CHANGE HISTORY

21308 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

cacos()*System Interfaces*21309 **NAME**21310 `cacos`, `cacosf`, `cacosl` — complex arc cosine functions21311 **SYNOPSIS**21312 `#include <complex.h>`21313 `double complex cacos(double complex z);`21314 `float complex cacosf(float complex z);`21315 `long double complex cacosl(long double complex z);`21316 **DESCRIPTION**

21317 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21318 conflict between the requirements described here and the ISO C standard is unintentional. This
 21319 volume of POSIX.1-2008 defers to the ISO C standard.

21320 These functions shall compute the complex arc cosine of z , with branch cuts outside the interval
 21321 $[-1, +1]$ along the real axis.

21322 **RETURN VALUE**

21323 These functions shall return the complex arc cosine value, in the range of a strip mathematically
 21324 unbounded along the imaginary axis and in the interval $[0, \pi]$ along the real axis.

21325 **ERRORS**

21326 No errors are defined.

21327 **EXAMPLES**

21328 None.

21329 **APPLICATION USAGE**

21330 None.

21331 **RATIONALE**

21332 None.

21333 **FUTURE DIRECTIONS**

21334 None.

21335 **SEE ALSO**21336 `ccos()`21337 XBD `<complex.h>`21338 **CHANGE HISTORY**

21339 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21340 **NAME**

21341 cacosh, cacoshf, cacoshl — complex arc hyperbolic cosine functions

21342 **SYNOPSIS**21343 `#include <complex.h>`21344 `double complex cacosh(double complex z);`21345 `float complex cacoshf(float complex z);`21346 `long double complex cacoshl(long double complex z);`21347 **DESCRIPTION**

21348 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21349 conflict between the requirements described here and the ISO C standard is unintentional. This
 21350 volume of POSIX.1-2008 defers to the ISO C standard.

21351 These functions shall compute the complex arc hyperbolic cosine of z , with a branch cut at
 21352 values less than 1 along the real axis.

21353 **RETURN VALUE**

21354 These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip
 21355 of non-negative values along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary axis.

21356 **ERRORS**

21357 No errors are defined.

21358 **EXAMPLES**

21359 None.

21360 **APPLICATION USAGE**

21361 None.

21362 **RATIONALE**

21363 None.

21364 **FUTURE DIRECTIONS**

21365 None.

21366 **SEE ALSO**21367 [ccosh\(\)](#)21368 XBD [<complex.h>](#)21369 **CHANGE HISTORY**

21370 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

cacosl()*System Interfaces*21371 **NAME**

21372 cacosl — complex arc cosine functions

21373 **SYNOPSIS**

21374 #include <complex.h>

21375 long double complex cacosl(long double complex z);

21376 **DESCRIPTION**21377 Refer to *acos()*.

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

21378 NAME

21379 `calloc` — a memory allocator

21380 SYNOPSIS

21381 `#include <stdlib.h>`

21382 `void *calloc(size_t nelem, size_t elsize);`

21383 DESCRIPTION

21384 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21385 conflict between the requirements described here and the ISO C standard is unintentional. This
 21386 volume of POSIX.1-2008 defers to the ISO C standard.

21387 The `calloc()` function shall allocate unused space for an array of *nelem* elements each of whose
 21388 size in bytes is *elsize*. The space shall be initialized to all bits 0.

21389 The order and contiguity of storage allocated by successive calls to `calloc()` is unspecified. The
 21390 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
 21391 a pointer to any type of object and then used to access such an object or an array of such objects
 21392 in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall
 21393 yield a pointer to an object disjoint from any other object. The pointer returned shall point to the
 21394 start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer
 21395 shall be returned. If the size of the space requested is 0, the behavior is implementation-defined:
 21396 the value returned shall be either a null pointer or a unique pointer.

21397 RETURN VALUE

21398 Upon successful completion with both *nelem* and *elsize* non-zero, `calloc()` shall return a pointer to
 21399 the allocated space. If either *nelem* or *elsize* is 0, then either a null pointer or a unique pointer
 21400 value that can be successfully passed to `free()` shall be returned. Otherwise, it shall return a null
 21401 CX pointer and set *errno* to indicate the error.

21402 ERRORS

21403 The `calloc()` function shall fail if:

21404 CX [ENOMEM] Insufficient memory is available.

21405 EXAMPLES

21406 None.

21407 APPLICATION USAGE

21408 There is now no requirement for the implementation to support the inclusion of `<malloc.h>`.

21409 RATIONALE

21410 None.

21411 FUTURE DIRECTIONS

21412 None.

21413 SEE ALSO

21414 [*free\(\)*](#), [*malloc\(\)*](#), [*realloc\(\)*](#)

21415 XBD `<stdlib.h>`

21416 CHANGE HISTORY

21417 First released in Issue 1. Derived from Issue 1 of the SVID.

21418 Issue 6

21419 Extensions beyond the ISO C standard are marked.

calloc()*System Interfaces*

- 21420 The following new requirements on POSIX implementations derive from alignment with the
21421 Single UNIX Specification:
- 21422 • The setting of *errno* and the [ENOMEM] error condition are mandatory if an insufficient
21423 memory condition occurs.

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

21424 NAME

21425 **carg, cargf, cargl** — complex argument functions

21426 SYNOPSIS

```
21427 #include <complex.h>
21428 double carg(double complex z);
21429 float cargf(float complex z);
21430 long double cargl(long double complex z);
```

21431 DESCRIPTION

21432 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 21433 conflict between the requirements described here and the ISO C standard is unintentional. This
 21434 volume of POSIX.1-2008 defers to the ISO C standard.

21435 These functions shall compute the argument (also called phase angle) of z , with a branch cut
 21436 along the negative real axis.

21437 RETURN VALUE

21438 These functions shall return the value of the argument in the interval $[-\pi, +\pi]$.

21439 ERRORS

21440 No errors are defined.

21441 EXAMPLES

21442 None.

21443 APPLICATION USAGE

21444 None.

21445 RATIONALE

21446 None.

21447 FUTURE DIRECTIONS

21448 None.

21449 SEE ALSO

21450 *cimag(), conj(), cproj()*

21451 XBD **<complex.h>**

21452 CHANGE HISTORY

21453 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

casin()*System Interfaces*21454 **NAME**21455 `casin, casinf, casinl` — complex arc sine functions21456 **SYNOPSIS**21457 `#include <complex.h>`21458 `double complex casin(double complex z);`21459 `float complex casinf(float complex z);`21460 `long double complex casinl(long double complex z);`21461 **DESCRIPTION**

21462 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21463 conflict between the requirements described here and the ISO C standard is unintentional. This
 21464 volume of POSIX.1-2008 defers to the ISO C standard.

21465 These functions shall compute the complex arc sine of z , with branch cuts outside the interval
 21466 $[-1, +1]$ along the real axis.

21467 **RETURN VALUE**

21468 These functions shall return the complex arc sine value, in the range of a strip mathematically
 21469 unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the real axis.

21470 **ERRORS**

21471 No errors are defined.

21472 **EXAMPLES**

21473 None.

21474 **APPLICATION USAGE**

21475 None.

21476 **RATIONALE**

21477 None.

21478 **FUTURE DIRECTIONS**

21479 None.

21480 **SEE ALSO**21481 [*csin\(\)*](#)21482 XBD [*<complex.h>*](#)21483 **CHANGE HISTORY**

21484 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21485 **NAME**21486 `casinh, casinhf, casinhl` — complex arc hyperbolic sine functions21487 **SYNOPSIS**21488 `#include <complex.h>`21489 `double complex casinh(double complex z);`21490 `float complex casinhf(float complex z);`21491 `long double complex casinhl(long double complex z);`21492 **DESCRIPTION**

21493 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21494 conflict between the requirements described here and the ISO C standard is unintentional. This
 21495 volume of POSIX.1-2008 defers to the ISO C standard.

21496 These functions shall compute the complex arc hyperbolic sine of z , with branch cuts outside the
 21497 interval $[-i, +i]$ along the imaginary axis.

21498 **RETURN VALUE**

21499 These functions shall return the complex arc hyperbolic sine value, in the range of a strip
 21500 mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the
 21501 imaginary axis.

21502 **ERRORS**

21503 No errors are defined.

21504 **EXAMPLES**

21505 None.

21506 **APPLICATION USAGE**

21507 None.

21508 **RATIONALE**

21509 None.

21510 **FUTURE DIRECTIONS**

21511 None.

21512 **SEE ALSO**21513 `csinh()`21514 XBD `<complex.h>`21515 **CHANGE HISTORY**

21516 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

casinl()*System Interfaces*21517 **NAME**21518 **casinl** — complex arc sine functions21519 **SYNOPSIS**21520 `#include <complex.h>`21521 `long double complex casinl(long double complex z);`21522 **DESCRIPTION**21523 Refer to *casin()*.

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

21524 NAME

21525 `catan`, `catanf`, `catanl` — complex arc tangent functions

21526 SYNOPSIS

21527 `#include <complex.h>`

21528 `double complex catan(double complex z);`
 21529 `float complex catanf(float complex z);`
 21530 `long double complex catanl(long double complex z);`

21531 DESCRIPTION

21532 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21533 conflict between the requirements described here and the ISO C standard is unintentional. This
 21534 volume of POSIX.1-2008 defers to the ISO C standard.

21535 These functions shall compute the complex arc tangent of z , with branch cuts outside the
 21536 interval $[-i, +i]$ along the imaginary axis.

21537 RETURN VALUE

21538 These functions shall return the complex arc tangent value, in the range of a strip
 21539 mathematically unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the
 21540 real axis.

21541 ERRORS

21542 No errors are defined.

21543 EXAMPLES

21544 None.

21545 APPLICATION USAGE

21546 None.

21547 RATIONALE

21548 None.

21549 FUTURE DIRECTIONS

21550 None.

21551 SEE ALSO

21552 [`ctan\(\)`](#)

21553 XBD [`<complex.h>`](#)

21554 CHANGE HISTORY

21555 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

catanh()*System Interfaces*21556 **NAME**

21557 catanh, catanhf, catanhl — complex arc hyperbolic tangent functions

21558 **SYNOPSIS**

21559 #include <complex.h>

21560 double complex catanh(double complex z);

21561 float complex catanhf(float complex z);

21562 long double complex catanhl(long double complex z);

21563 **DESCRIPTION**

21564 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21565 conflict between the requirements described here and the ISO C standard is unintentional. This
 21566 volume of POSIX.1-2008 defers to the ISO C standard.

21567 These functions shall compute the complex arc hyperbolic tangent of z , with branch cuts outside
 21568 the interval $[-1, +1]$ along the real axis.

21569 **RETURN VALUE**

21570 These functions shall return the complex arc hyperbolic tangent value, in the range of a strip
 21571 mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the
 21572 imaginary axis.

21573 **ERRORS**

21574 No errors are defined.

21575 **EXAMPLES**

21576 None.

21577 **APPLICATION USAGE**

21578 None.

21579 **RATIONALE**

21580 None.

21581 **FUTURE DIRECTIONS**

21582 None.

21583 **SEE ALSO**

21584 ctanh()

21585 XBD <complex.h>

21586 **CHANGE HISTORY**

21587 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

*System Interfaces***catanl()**21588 **NAME**

21589 catanl — complex arc tangent functions

21590 **SYNOPSIS**

21591 #include <complex.h>

21592 long double complex catanl(long double complex z);

21593 **DESCRIPTION**21594 Refer to *catan()*.

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

catclose()*System Interfaces*21595 **NAME**21596 **catclose** — close a message catalog descriptor21597 **SYNOPSIS**21598 `#include <nl_types.h>`21599 `int catclose(nl_catd catd);`21600 **DESCRIPTION**

21601 The *catclose()* function shall close the message catalog identified by *catd*. If a file descriptor is
 21602 used to implement the type **nl_catd**, that file descriptor shall be closed.

21603 **RETURN VALUE**

21604 Upon successful completion, *catclose()* shall return 0; otherwise, -1 shall be returned, and *errno*
 21605 set to indicate the error.

21606 **ERRORS**21607 The *catclose()* function may fail if:

21608 [EBADF] The catalog descriptor is not valid.

21609 [EINTR] The *catclose()* function was interrupted by a signal.21610 **EXAMPLES**

21611 None.

21612 **APPLICATION USAGE**

21613 None.

21614 **RATIONALE**

21615 None.

21616 **FUTURE DIRECTIONS**

21617 None.

21618 **SEE ALSO**21619 *catgets()*, *catopen()*21620 XBD *<nl_types.h>*21621 **CHANGE HISTORY**

21622 First released in Issue 2.

21623 **Issue 7**21624 The *catclose()* function is moved from the XSI option to the Base.

System Interfaces

catgets()**NAME**

catgets — read a program message

SYNOPSIS

```
#include <nl_types.h>
```

```
char *catgets(nl_catd catd, int set_id, int msg_id, const char *s);
```

DESCRIPTION

The *catgets()* function shall attempt to read message *msg_id*, in set *set_id*, from the message catalog identified by *catd*. The *catd* argument is a message catalog descriptor returned from an earlier call to *catopen()*. The results are undefined if *catd* is not a value returned by *catopen()* for a message catalog still open in the process. The *s* argument points to a default message string which shall be returned by *catgets()* if it cannot retrieve the identified message.

The *catgets()* function need not be thread-safe.

RETURN VALUE

If the identified message is retrieved successfully, *catgets()* shall return a pointer to an internal buffer area containing the null-terminated message string. If the call is unsuccessful for any reason, *s* shall be returned and *errno* shall be set to indicate the error.

ERRORS

The *catgets()* function shall fail if:

[EINTR] The read operation was terminated due to the receipt of a signal, and no data was transferred.

[ENOMSG] The message identified by *set_id* and *msg_id* is not in the message catalog.

The *catgets()* function may fail if:

[EBADF] The *catd* argument is not a valid message catalog descriptor open for reading.

[EBADMSG] The message identified by *set_id* and *msg_id* in the specified message catalog did not satisfy implementation-defined security criteria.

[EINVAL] The message catalog identified by *catd* is corrupted.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catclose(), *catopen()*

XBD *<nl_types.h>*

CHANGE HISTORY

First released in Issue 2.

catgets()*System Interfaces*21664 **Issue 5**

21665 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

21666 **Issue 6**

21667 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

21668 **Issue 7**

21669 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [EINTR] and
21670 [ENOMSG] errors to become “shall fail” errors, updating the RETURN VALUE section, and
21671 updating the DESCRIPTION to note that: “The results are undefined if *catd* is not a value
21672 returned by *catopen()* for a message catalog still open in the process.

21673 Austin Group Interpretation 1003.1-2001 #148 is applied, adding

21674 The *catgets()* function is moved from the XSI option to the Base.

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

System Interfaces

catopen()**NAME**

catopen — open a message catalog

SYNOPSIS

```
#include <nl_types.h>

nl_catd catopen(const char *name, int oflag);
```

DESCRIPTION

The *catopen()* function shall open a message catalog and return a message catalog descriptor. The *name* argument specifies the name of the message catalog to be opened. If *name* contains a *'/'*, then *name* specifies a complete name for the message catalog. Otherwise, the environment variable *NLSPATH* is used with *name* substituted for the *%N* conversion specification (see XBD Chapter 8, on page 173). If *NLSPATH* exists in the environment when the process starts, then if the process has appropriate privileges, the behavior of *catopen()* is undefined. If *NLSPATH* does not exist in the environment, or if a message catalog cannot be found in any of the components specified by *NLSPATH*, then an implementation-defined default path shall be used. This default may be affected by the setting of *LC_MESSAGES* if the value of *oflag* is *NL_CAT_LOCALE*, or the *LANG* environment variable if *oflag* is 0.

A message catalog descriptor shall remain valid in a process until that process closes it, or a successful call to one of the *exec* functions. A change in the setting of the *LC_MESSAGES* category may invalidate existing open catalogs.

If a file descriptor is used to implement message catalog descriptors, the *FD_CLOEXEC* flag shall be set; see <fcntl.h>.

If the value of the *oflag* argument is 0, the *LANG* environment variable is used to locate the catalog without regard to the *LC_MESSAGES* category. If the *oflag* argument is *NL_CAT_LOCALE*, the *LC_MESSAGES* category is used to locate the message catalog (see XBD Section 8.2, on page 174).

RETURN VALUE

Upon successful completion, *catopen()* shall return a message catalog descriptor for use on subsequent calls to *catgets()* and *catclose()*. Otherwise, *catopen()* shall return (*nl_catd*) -1 and set *errno* to indicate the error.

ERRORS

The *catopen()* function may fail if:

[EACCES] Search permission is denied for the component of the path prefix of the message catalog or read permission is denied for the message catalog.

[EMFILE] All file descriptors available to the process are currently open.

[ENAMETOOLONG]

The length of a component of a pathname is longer than {NAME_MAX}.

[ENAMETOOLONG]

The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[ENFILE] Too many files are currently open in the system.

[ENOENT] The message catalog does not exist or the *name* argument points to an empty string.

catopen()*System Interfaces*

21718	[ENOMEM]	Insufficient storage space is available.
21719	[ENOTDIR]	A component of the path prefix of the message catalog is not a directory, or the pathname of the message catalog contains at least one non- <code><slash></code> character and ends with one or more trailing <code><slash></code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
21720		
21721		
21722		
21723		

EXAMPLES

21724
21725 None.

APPLICATION USAGE

21726 Some implementations of *catopen()* use *malloc()* to allocate space for internal buffer areas. The
21727 *catopen()* function may fail if there is insufficient storage space available to accommodate these
21728 buffers.
21729

21730 Conforming applications must assume that message catalog descriptors are not valid after a call
21731 to one of the *exec* functions.

21732 Application developers should be aware that guidelines for the location of message catalogs
21733 have not yet been developed. Therefore they should take care to avoid conflicting with catalogs
21734 used by other applications and the standard utilities.

RATIONALE

21735
21736 None.

FUTURE DIRECTIONS

21737
21738 None.

SEE ALSO

21740 *catclose()*, *catgets()*
21741 XBD Chapter 8 (on page 173), `<fcntl.h>`, `<nl_types.h>`,

CHANGE HISTORY

21742 First released in Issue 2.

Issue 7

21744 Austin Group Interpretation 1003.1-2001 #143 is applied.
21745
21746 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
21747 The *catopen()* function is moved from the XSI option to the Base.

21748 **NAME**

21749 cbrt, cbrtf, cbrtl — cube root functions

21750 **SYNOPSIS**

```
21751 #include <math.h>
21752 double cbrt(double x);
21753 float cbrtf(float x);
21754 long double cbrtl(long double x);
```

21755 **DESCRIPTION**

21756 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21757 conflict between the requirements described here and the ISO C standard is unintentional. This
 21758 volume of POSIX.1-2008 defers to the ISO C standard.

21759 These functions shall compute the real cube root of their argument x .

21760 **RETURN VALUE**

21761 Upon successful completion, these functions shall return the cube root of x .

21762 MX If x is NaN, a NaN shall be returned.

21763 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

21764 **ERRORS**

21765 No errors are defined.

21766 **EXAMPLES**

21767 None.

21768 **APPLICATION USAGE**

21769 None.

21770 **RATIONALE**

21771 For some applications, a true cube root function, which returns negative results for negative
 21772 arguments, is more appropriate than $\text{pow}(x, 1.0/3.0)$, which returns a NaN for x less than 0.

21773 **FUTURE DIRECTIONS**

21774 None.

21775 **SEE ALSO**

21776 XBD [<math.h>](#)

21777 **CHANGE HISTORY**

21778 First released in Issue 4, Version 2.

21779 **Issue 5**

21780 Moved from X/OPEN UNIX extension to BASE.

21781 **Issue 6**

21782 The `cbrt()` function is no longer marked as an extension.

21783 The `cbrtf()` and `cbrtl()` functions are added for alignment with the ISO/IEC 9899:1999 standard.

21784 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 21785 revised to align with the ISO/IEC 9899:1999 standard.

21786 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 21787 marked.

ccos()*System Interfaces*21788 **NAME**

21789 ccos, ccosh, ccoshl — complex cosine functions

21790 **SYNOPSIS**

21791 #include <complex.h>

21792 double complex ccos(double complex z);

21793 float complex ccosh(float complex z);

21794 long double complex ccoshl(long double complex z);

21795 **DESCRIPTION**

21796 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21797 conflict between the requirements described here and the ISO C standard is unintentional. This
 21798 volume of POSIX.1-2008 defers to the ISO C standard.

21799 These functions shall compute the complex cosine of z.

21800 **RETURN VALUE**

21801 These functions shall return the complex cosine value.

21802 **ERRORS**

21803 No errors are defined.

21804 **EXAMPLES**

21805 None.

21806 **APPLICATION USAGE**

21807 None.

21808 **RATIONALE**

21809 None.

21810 **FUTURE DIRECTIONS**

21811 None.

21812 **SEE ALSO**21813 [ccosh\(\)](#)21814 XBD [<complex.h>](#)21815 **CHANGE HISTORY**

21816 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

System Interfaces

ccosh()21817 **NAME**21818 `ccosh, ccoshf, ccoshl` — complex hyperbolic cosine functions21819 **SYNOPSIS**21820 `#include <complex.h>`21821 `double complex ccosh(double complex z);`21822 `float complex ccoshf(float complex z);`21823 `long double complex ccoshl(long double complex z);`21824 **DESCRIPTION**

21825 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21826 conflict between the requirements described here and the ISO C standard is unintentional. This
 21827 volume of POSIX.1-2008 defers to the ISO C standard.

21828 These functions shall compute the complex hyperbolic cosine of *z*.21829 **RETURN VALUE**

21830 These functions shall return the complex hyperbolic cosine value.

21831 **ERRORS**

21832 No errors are defined.

21833 **EXAMPLES**

21834 None.

21835 **APPLICATION USAGE**

21836 None.

21837 **RATIONALE**

21838 None.

21839 **FUTURE DIRECTIONS**

21840 None.

21841 **SEE ALSO**21842 [*cacosh\(\)*](#)21843 XBD [*<complex.h>*](#)21844 **CHANGE HISTORY**

21845 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

ccosl()*System Interfaces*21846 **NAME**

21847 ccosl — complex cosine functions

21848 **SYNOPSIS**

21849 #include <complex.h>

21850 long double complex ccosl(long double complex z);

21851 **DESCRIPTION**21852 Refer to *ccos()*.

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

21853 NAME

21854 ceil, ceilf, ceill — ceiling value function

21855 SYNOPSIS

```
21856 #include <math.h>
21857 double ceil(double x);
21858 float ceilf(float x);
21859 long double ceill(long double x);
```

21860 DESCRIPTION

21861 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21862 conflict between the requirements described here and the ISO C standard is unintentional. This
 21863 volume of POSIX.1-2008 defers to the ISO C standard.

21864 These functions shall compute the smallest integral value not less than *x*.

21865 An application wishing to check for error situations should set *errno* to zero and call
 21866 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 21867 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 21868 zero, an error has occurred.

21869 RETURN VALUE

21870 Upon successful completion, *ceil()*, *ceilf()*, and *ceill()* shall return the smallest integral value not
 21871 less than *x*, expressed as a type **double**, **float**, or **long double**, respectively.

21872 MX If *x* is NaN, a NaN shall be returned.

21873 If *x* is ± 0 or $\pm \text{Inf}$, *x* shall be returned.

21874 XSI If the correct value would cause overflow, a range error shall occur and *ceil()*, *ceilf()*, and *ceill()*
 21875 shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

21876 ERRORS

21877 These functions shall fail if:

21878 XSI Range Error The result overflows.

21879 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 21880 then *errno* shall be set to [ERANGE]. If the integer expression
 21881 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 21882 floating-point exception shall be raised.

21883 EXAMPLES

21884 None.

21885 APPLICATION USAGE

21886 The integral value returned by these functions need not be expressible as an **int** or **long**. The
 21887 return value should be tested before assigning it to an integer type to avoid the undefined
 21888 results of an integer overflow.

21889 The *ceil()* function can only overflow when the floating-point representation has
 21890 DBL_MANT_DIG > DBL_MAX_EXP.

21891 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 21892 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

ceil()*System Interfaces*21893 **RATIONALE**

21894 None.

21895 **FUTURE DIRECTIONS**

21896 None.

21897 **SEE ALSO**21898 *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*21899 XBD [Section 4.19](#) (on page 116), [<math.h>](#)21900 **CHANGE HISTORY**

21901 First released in Issue 1. Derived from Issue 1 of the SVID.

21902 **Issue 5**21903 The DESCRIPTION is updated to indicate how an application should check for an error. This
21904 text was previously published in the APPLICATION USAGE section.21905 **Issue 6**21906 The *ceilf()* and *ceil()* functions are added for alignment with the ISO/IEC 9899:1999 standard.21907 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
21908 revised to align with the ISO/IEC 9899:1999 standard.21909 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
21910 marked.

21911 **NAME**

21912 cexp, cexpf, cexpl — complex exponential functions

21913 **SYNOPSIS**

21914 #include <complex.h>

21915 double complex cexp(double complex z);

21916 float complex cexpf(float complex z);

21917 long double complex cexpl(long double complex z);

21918 **DESCRIPTION**

21919 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21920 conflict between the requirements described here and the ISO C standard is unintentional. This
 21921 volume of POSIX.1-2008 defers to the ISO C standard.

21922 These functions shall compute the complex exponent of z , defined as e^z .21923 **RETURN VALUE**21924 These functions shall return the complex exponential value of z .21925 **ERRORS**

21926 No errors are defined.

21927 **EXAMPLES**

21928 None.

21929 **APPLICATION USAGE**

21930 None.

21931 **RATIONALE**

21932 None.

21933 **FUTURE DIRECTIONS**

21934 None.

21935 **SEE ALSO**21936 [clog\(\)](#)21937 XBD [<complex.h>](#)21938 **CHANGE HISTORY**

21939 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

cfgetispeed()

System Interfaces

21940 **NAME**21941 `cfgetispeed` — get input baud rate21942 **SYNOPSIS**21943 `#include <termios.h>`21944 `speed_t cfgetispeed(const struct termios *termios_p);`21945 **DESCRIPTION**21946 The `cfgetispeed()` function shall extract the input baud rate from the **termios** structure to which
21947 the `termios_p` argument points.21948 This function shall return exactly the value in the **termios** data structure, without interpretation.21949 **RETURN VALUE**21950 Upon successful completion, `cfgetispeed()` shall return a value of type **speed_t** representing the
21951 input baud rate.21952 **ERRORS**

21953 No errors are defined.

21954 **EXAMPLES**

21955 None.

21956 **APPLICATION USAGE**

21957 None.

21958 **RATIONALE**21959 The term “baud” is used historically here, but is not technically correct. This is properly “bits per
21960 second”, which may not be the same as **baud**. However, the term is used because of the
21961 historical usage and understanding.21962 The `cfgetispeed()`, `cfgetispeed()`, `cfsetispeed()`, and `cfsetispeed()` functions do not take arguments as
21963 numbers, but rather as symbolic names. There are two reasons for this:

- 21964 1. Historically, numbers were not used because of the way the rate was stored in the data
21965 structure. This is retained even though a function is now used.
- 21966 2. More importantly, only a limited set of possible rates is at all portable, and this constrains
21967 the application to that set.

21968 There is nothing to prevent an implementation accepting as an extension a number (such as 126),
21969 and since the encoding of the Bxxx symbols is not specified, this can be done to avoid
21970 introducing ambiguity.

21971 Setting the input baud rate to zero was a mechanism to allow for split baud rates. Clarifications
21972 in this volume of POSIX.1-2008 have made it possible to determine whether split rates are
21973 supported and to support them without having to treat zero as a special case. Since this
21974 functionality is also confusing, it has been declared obsolescent. The 0 argument referred to is
21975 the literal constant 0, not the symbolic constant B0. This volume of POSIX.1-2008 does not
21976 preclude B0 from being defined as the value 0; in fact, implementations would likely benefit
21977 from the two being equivalent. This volume of POSIX.1-2008 does not fully specify whether the
21978 previous `cfsetispeed()` value is retained after a `tcgetattr()` as the actual value or as zero. Therefore,
21979 conforming applications should always set both the input speed and output speed when setting
21980 either.

21981 In historical implementations, the baud rate information is traditionally kept in **c_cflag**.
21982 Applications should be written to presume that this might be the case (and thus not blindly copy
21983 **c_cflag**), but not to rely on it in case it is in some other field of the structure. Setting the **c_cflag**
21984 field absolutely after setting a baud rate is a non-portable action because of this. In general, the

21985 unused parts of the flag fields might be used by the implementation and should not be blindly
21986 copied from the descriptions of one terminal device to another.

21987 **FUTURE DIRECTIONS**

21988 None.

21989 **SEE ALSO**

21990 [*cfgetospeed\(\)*](#), [*cfsetispeed\(\)*](#), [*cfsetospeed\(\)*](#), [*tcgetattr\(\)*](#)

21991 XBD [Chapter 11](#) (on page 199), [**<termios.h>**](#)

21992 **CHANGE HISTORY**

21993 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

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

cfgetospeed()

System Interfaces

21994 **NAME**21995 **cfgetospeed** — get output baud rate21996 **SYNOPSIS**21997 `#include <termios.h>`21998 `speed_t cfgetospeed(const struct termios *termios_p);`21999 **DESCRIPTION**22000 The *cfgetospeed()* function shall extract the output baud rate from the **termios** structure to which
22001 the *termios_p* argument points.22002 This function shall return exactly the value in the **termios** data structure, without interpretation.22003 **RETURN VALUE**22004 Upon successful completion, *cfgetospeed()* shall return a value of type **speed_t** representing the
22005 output baud rate.22006 **ERRORS**

22007 No errors are defined.

22008 **EXAMPLES**

22009 None.

22010 **APPLICATION USAGE**

22011 None.

22012 **RATIONALE**22013 Refer to *cfgetispeed()*.22014 **FUTURE DIRECTIONS**

22015 None.

22016 **SEE ALSO**22017 *cfgetispeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*22018 XBD Chapter 11 (on page 199), **<termios.h>**22019 **CHANGE HISTORY**

22020 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

NAME

cfsetispeed — set input baud rate

SYNOPSIS

```
#include <termios.h>
```

```
int cfsetispeed(struct termios *termios_p, speed_t speed);
```

DESCRIPTION

The *cfsetispeed()* function shall set the input baud rate stored in the structure pointed to by *termios_p* to *speed*.

There shall be no effect on the baud rates set in the hardware until a subsequent successful call to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set baud rates not supported by the terminal device need not be detected until the *tcsetattr()* function is called.

RETURN VALUE

Upon successful completion, *cfsetispeed()* shall return 0; otherwise, -1 shall be returned, and *errno* may be set to indicate the error.

ERRORS

The *cfsetispeed()* function may fail if:

[EINVAL] The *speed* value is not a valid baud rate.

[EINVAL] The value of *speed* is outside the range of possible speed values as specified in **<termios.h>**.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *cfgetispeed()*.

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetispeed(), *cfgetospeed()*, *cfsetospeed()*, *tcsetattr()*

XBD Chapter 11 (on page 199), **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional setting of *errno* and the [EINVAL] error conditions are added.

cfsetospeed()

System Interfaces

22058 **NAME**22059 **cfsetospeed** — set output baud rate22060 **SYNOPSIS**22061 `#include <termios.h>`22062 `int cfsetospeed(struct termios *termios_p, speed_t speed);`22063 **DESCRIPTION**22064 The *cfsetospeed()* function shall set the output baud rate stored in the structure pointed to by
22065 *termios_p* to *speed*.22066 There shall be no effect on the baud rates set in the hardware until a subsequent successful call
22067 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set
22068 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*
22069 function is called.22070 **RETURN VALUE**22071 Upon successful completion, *cfsetospeed()* shall return 0; otherwise, it shall return -1 and *errno*
22072 may be set to indicate the error.22073 **ERRORS**22074 The *cfsetospeed()* function may fail if:22075 [EINVAL] The *speed* value is not a valid baud rate.22076 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in
22077 **<termios.h>**.22078 **EXAMPLES**

22079 None.

22080 **APPLICATION USAGE**

22081 None.

22082 **RATIONALE**22083 Refer to *cfgetispeed()*.22084 **FUTURE DIRECTIONS**

22085 None.

22086 **SEE ALSO**22087 *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *tcsetattr()*22088 XBD Chapter 11 (on page 199), **<termios.h>**22089 **CHANGE HISTORY**

22090 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

22091 **Issue 6**22092 The following new requirements on POSIX implementations derive from alignment with the
22093 Single UNIX Specification:

- 22094
- The optional setting of *errno* and the [EINVAL] error conditions are added.

22095 NAME

22096 **chdir** — change working directory

22097 SYNOPSIS

22098 `#include <unistd.h>`
 22099 `int chdir(const char *path);`

22100 DESCRIPTION

22101 The *chdir()* function shall cause the directory named by the pathname pointed to by the *path*
 22102 argument to become the current working directory; that is, the starting point for path searches
 22103 for pathnames not beginning with `'/'`.

22104 RETURN VALUE

22105 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, the current
 22106 working directory shall remain unchanged, and *errno* shall be set to indicate the error.

22107 ERRORS

22108 The *chdir()* function shall fail if:

- 22109 [EACCES] Search permission is denied for any component of the pathname.
 - 22110 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 22111 argument.
 - 22112 [ENAMETOOLONG]
 22113 The length of a component of a pathname is longer than {NAME_MAX}.
 - 22114 [ENOENT] A component of *path* does not name an existing directory or *path* is an empty
 22115 string.
 - 22116 [ENOTDIR] A component of the pathname is not a directory.
- 22117 The *chdir()* function may fail if:
- 22118 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 22119 resolution of the *path* argument.
 - 22120 [ENAMETOOLONG]
 22121 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 22122 symbolic link produced an intermediate result with a length that exceeds
 22123 {PATH_MAX}.

22124 EXAMPLES**22125 Changing the Current Working Directory**

22126 The following example makes the value pointed to by **directory**, `/tmp`, the current working
 22127 directory.

22128 `#include <unistd.h>`
 22129 `...`
 22130 `char *directory = "/tmp";`
 22131 `int ret;`
 22132 `ret = chdir (directory);`

chdir()*System Interfaces*22133 **APPLICATION USAGE**

22134 None.

22135 **RATIONALE**22136 The *chdir()* function only affects the working directory of the current process.22137 **FUTURE DIRECTIONS**

22138 None.

22139 **SEE ALSO**22140 *getcwd()*22141 XBD **<unistd.h>**22142 **CHANGE HISTORY**

22143 First released in Issue 1. Derived from Issue 1 of the SVID.

22144 **Issue 6**

22145 The APPLICATION USAGE section is added.

22146 The following new requirements on POSIX implementations derive from alignment with the
22147 Single UNIX Specification:

- 22148 • The [ELOOP] mandatory error condition is added.
- 22149 • A second [ENAMETOOLONG] is added as an optional error condition.

22150 The following changes were made to align with the IEEE P1003.1a draft standard:

- 22151 • The [ELOOP] optional error condition is added.

22152 **Issue 7**

22153 Austin Group Interpretation 1003.1-2001-#143 is applied.

System Interfaces

chmod()**NAME**

chmod, fchmodat — change mode of a file relative to directory file descriptor

SYNOPSIS

```
#include <sys/stat.h>
```

```
int chmod(const char *path, mode_t mode);
```

```
int fchmodat(int fd, const char *path, mode_t mode, int flag);
```

DESCRIPTION

The *chmod()* function shall change S_ISUID, S_ISGID, S_ISVTX, and the file permission bits of the file named by the pathname pointed to by the *path* argument to the corresponding bits in the *mode* argument. The application shall ensure that the effective user ID of the process matches the owner of the file or the process has appropriate privileges in order to do this.

S_ISUID, S_ISGID, S_ISVTX, and the file permission bits are described in <sys/stat.h>.

If the calling process does not have appropriate privileges, and if the group ID of the file does not match the effective group ID or one of the supplementary group IDs and if the file is a regular file, bit S_ISGID (set-group-ID on execution) in the file's mode shall be cleared upon successful return from *chmod()*.

Additional implementation-defined restrictions may cause the S_ISUID and S_ISGID bits in *mode* to be ignored.

The effect on file descriptors for files open at the time of a call to *chmod()* is implementation-defined.

Upon successful completion, *chmod()* shall mark for update the last file status change timestamp of the file.

The *fchmodat()* function shall be equivalent to the *chmod()* function except in the case where *path* specifies a relative path. In this case the file to be changed is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function shall not perform the check.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in <fcntl.h>:

AT_SYMLINK_NOFOLLOW

If *path* names a symbolic link, then the mode of the symbolic link is changed.

If *fchmodat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working directory is used. If also *flag* is zero, the behavior shall be identical to a call to *chmod()*.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return -1 and set *errno* to indicate the error. If -1 is returned, no change to the file mode occurs.

ERRORS

These functions shall fail if:

[EACCES] Search permission is denied on a component of the path prefix.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

chmod()*System Interfaces*

22196	[ENAMETOOLONG]	
22197		The length of a component of a pathname is longer than {NAME_MAX}.
22198	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
22199	[ENOTDIR]	A component of the path prefix is not a directory, or the <i>path</i> argument
22200		contains at least one non- <code><slash></code> character and ends with one or more trailing
22201		<code><slash></code> characters and the last pathname component names an existing file
22202		that is neither a directory nor a symbolic link to a directory.
22203	[EPERM]	The effective user ID does not match the owner of the file and the process does
22204		not have appropriate privileges.
22205	[EROFS]	The named file resides on a read-only file system.
22206	The <i>fchmodat()</i> function shall fail if:	
22207	[EACCES]	<i>fd</i> was not opened with <code>O_SEARCH</code> and the permissions of the directory
22208		underlying <i>fd</i> do not permit directory searches.
22209	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is
22210		neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching.
22211	These functions may fail if:	
22212	[EINTR]	A signal was caught during execution of the function.
22213	[EINVAL]	The value of the <i>mode</i> argument is invalid.
22214	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
22215		resolution of the <i>path</i> argument.
22216	[ENAMETOOLONG]	
22217		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
22218		symbolic link produced an intermediate result with a length that exceeds
22219		{PATH_MAX}.
22220	The <i>fchmodat()</i> function may fail if:	
22221	[EINVAL]	The value of the <i>flag</i> argument is invalid.
22222	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is neither <code>AT_FDCWD</code> nor a
22223		file descriptor associated with a directory.
22224	[EOPNOTSUPP]	The <code>AT_SYMLINK_NOFOLLOW</code> bit is set in the <i>flag</i> argument, <i>path</i> names a
22225		symbolic link, and the system does not support changing the mode of a
22226		symbolic link.

EXAMPLES**Setting Read Permissions for User, Group, and Others**

The following example sets read permissions for the owner, group, and others.

```
#include <sys/stat.h>

const char *path;
...
chmod(path, S_IRUSR | S_IRGRP | S_IROTH);
```

Setting Read, Write, and Execute Permissions for the Owner Only

The following example sets read, write, and execute permissions for the owner, and no permissions for group and others.

```
#include <sys/stat.h>

const char *path;
...
chmod(path, S_IRWXU);
```

Setting Different Permissions for Owner, Group, and Other

The following example sets owner permissions for `CHANGEFILE` to read, write, and execute, group permissions to read and execute, and other permissions to read.

```
#include <sys/stat.h>

#define CHANGEFILE "/etc/myfile"
...
chmod(CHANGEFILE, S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
```

Setting and Checking File Permissions

The following example sets the file permission bits for a file named `/home/cnd/mod1`, then calls the `stat()` function to verify the permissions.

```
#include <sys/types.h>
#include <sys/stat.h>

int status;
struct stat buffer
...
chmod("home/cnd/mod1", S_IRWXU|S_IRWXG|S_IROTH|S_IWOTH);
status = stat("home/cnd/mod1", &buffer);
```

APPLICATION USAGE

In order to ensure that the `S_ISUID` and `S_ISGID` bits are set, an application requiring this should use `stat()` after a successful `chmod()` to verify this.

Any file descriptors currently open by any process on the file could possibly become invalid if the mode of the file is changed to a value which would deny access to that process. One situation where this could occur is on a stateless file system. This behavior will not occur in a conforming environment.

RATIONALE

This volume of POSIX.1-2008 specifies that the `S_ISGID` bit is cleared by `chmod()` on a regular file under certain conditions. This is specified on the assumption that regular files may be executed, and the system should prevent users from making executable `setgid()` files perform with privileges that the caller does not have. On implementations that support execution of other file types, the `S_ISGID` bit should be cleared for those file types under the same circumstances.

Implementations that use the `S_ISUID` bit to indicate some other function (for example, mandatory record locking) on non-executable files need not clear this bit on writing. They should clear the bit for executable files and any other cases where the bit grants special powers to processes that change the file contents. Similar comments apply to the `S_ISGID` bit.

The purpose of the `fchmodat()` function is to enable changing the mode of files in directories

chmod()*System Interfaces*

other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *chmod()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *fchmodat()* function it can be guaranteed that the changed file is located relative to the desired directory. Some implementations might allow changing the mode of symbolic links. This is not supported by the interfaces in the POSIX specification. Systems with such support provide an interface named *lchmod()*. To support such implementations *fchmodat()* has a *flag* parameter.

FUTURE DIRECTIONS

None.

SEE ALSO

access(), *chown()*, *exec*, *fstatat()*, *fstatvfs()*, *mkdir()*, *mkfifo()*, *mknod()*, *open()*

XBD [*<fcntl.h>*](#), [*<sys/stat.h>*](#), [*<sys/types.h>*](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include [*<sys/types.h>*](#) has been removed. Although [*<sys/types.h>*](#) was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EINVAL] and [EINTR] optional error conditions are added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

The normative text is updated to avoid use of the term “must” for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The *fchmodat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

NAME

chown, fchownat — change owner and group of a file relative to directory file descriptor

SYNOPSIS

```
#include <unistd.h>

int chown(const char *path, uid_t owner, gid_t group);
int fchownat(int fd, const char *path, uid_t owner, gid_t group,
             int flag);
```

DESCRIPTION

The *chown()* function shall change the user and group ownership of a file.

The *path* argument points to a pathname naming a file. The user ID and group ID of the named file shall be set to the numeric values contained in *owner* and *group*, respectively.

Only processes with an effective user ID equal to the user ID of the file or with appropriate privileges may change the ownership of a file. If `_POSIX_CHOWN_RESTRICTED` is in effect for *path*:

- Changing the user ID is restricted to processes with appropriate privileges.
- Changing the group ID is permitted to a process with an effective user ID equal to the user ID of the file, but without appropriate privileges, if and only if *owner* is equal to the file's user ID or `(uid_t)-1` and *group* is equal either to the calling process' effective group ID or to one of its supplementary group IDs.

If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of the file mode are set, and the process does not have appropriate privileges, the set-user-ID (`S_ISUID`) and set-group-ID (`S_ISGID`) bits of the file mode shall be cleared upon successful return from *chown()*. If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of the file mode are set, and the process has appropriate privileges, it is implementation-defined whether the set-user-ID and set-group-ID bits are altered. If the *chown()* function is successfully invoked on a file that is not a regular file and one or more of the `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of the file mode are set, the set-user-ID and set-group-ID bits may be cleared.

If *owner* or *group* is specified as `(uid_t)-1` or `(gid_t)-1`, respectively, the corresponding ID of the file shall not be changed. If both *owner* and *group* are `-1`, the times need not be updated.

Upon successful completion, *chown()* shall mark for update the last file status change timestamp of the file.

The *fchownat()* function shall be equivalent to the *chown()* and *lchown()* functions except in the case where *path* specifies a relative path. In this case the file to be changed is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without `O_SEARCH`, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function shall not perform the check.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in `<fcntl.h>`:

`AT_SYMLINK_NOFOLLOW`

If *path* names a symbolic link, ownership of the symbolic link is changed.

If *fchownat()* is passed the special value `AT_FDCWD` in the *fd* parameter, the current working directory is used and the behavior shall be identical to a call to *chown()* or *lchown()* respectively,

chown()*System Interfaces*

22354 depending on whether or not the AT_SYMLINK_NOFOLLOW bit is set in the *flag* argument.

RETURN VALUE

22356 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 22357 return -1 and set *errno* to indicate the error. If -1 is returned, no changes are made in the user ID
 22358 and group ID of the file.

ERRORS

22359 These functions shall fail if:

- 22360 [EACCES] Search permission is denied on a component of the path prefix.
- 22361 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 22362 argument.
- 22363 [ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.
- 22364 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 22365 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument
 22366 contains at least one non-*<slash>* character and ends with one or more trailing
 22367 *<slash>* characters and the last pathname component names an existing file
 22368 that is neither a directory nor a symbolic link to a directory.
- 22369 [EPERM] The effective user ID does not match the owner of the file, or the calling
 22370 process does not have appropriate privileges and _POSIX_CHOWN_RESTRICTED indicates that such privilege is required.

22371 [EROFS] The named file resides on a read-only file system.

22372 The *fchownat()* function shall fail if:

- 22373 [EACCES] *fd* was not opened with O_SEARCH and the permissions of the directory
 22374 underlying *fd* do not permit directory searches.
- 22375 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
 22376 neither AT_FDCWD nor a valid file descriptor open for reading or searching.

22377 These functions may fail if:

- 22378 [EIO] An I/O error occurred while reading or writing to the file system.
- 22379 [EINTR] The *chown()* function was interrupted by a signal which was caught.
- 22380 [EINVAL] The owner or group ID supplied is not a value supported by the
 22381 implementation.
- 22382 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 22383 resolution of the *path* argument.
- 22384 [ENAMETOOLONG] The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 22385 symbolic link produced an intermediate result with a length that exceeds
 22386 {PATH_MAX}.

22387 The *fchownat()* function may fail if:

- 22388 [EINVAL] The value of the *flag* argument is not valid.

22393 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
 22394 file descriptor associated with a directory.

22395 EXAMPLES

22396 None.

22397 APPLICATION USAGE

22398 Although *chown()* can be used on some implementations by the file owner to change the owner
 22399 and group to any desired values, the only portable use of this function is to change the group of
 22400 a file to the effective GID of the calling process or to a member of its group set.

22401 RATIONALE

22402 System III and System V allow a user to give away files; that is, the owner of a file may change
 22403 its user ID to anything. This is a serious problem for implementations that are intended to meet
 22404 government security regulations. Version 7 and 4.3 BSD permit only the superuser to change the
 22405 user ID of a file. Some government agencies (usually not ones concerned directly with security)
 22406 find this limitation too confining. This volume of POSIX.1-2008 uses *may* to permit secure
 22407 implementations while not disallowing System V.

22408 System III and System V allow the owner of a file to change the group ID to anything. Version 7
 22409 permits only the superuser to change the group ID of a file. 4.3 BSD permits the owner to
 22410 change the group ID of a file to its effective group ID or to any of the groups in the list of
 22411 supplementary group IDs, but to no others.

22412 The POSIX.1-1990 standard requires that the *chown()* function invoked by a non-appropriate
 22413 privileged process clear the S_ISGID and the S_ISUID bits for regular files, and permits them to
 22414 be cleared for other types of files. This is so that changes in accessibility do not accidentally
 22415 cause files to become security holes. Unfortunately, requiring these bits to be cleared on non-
 22416 executable data files also clears the mandatory file locking bit (shared with S_ISGID), which is
 22417 an extension on many implementations (it first appeared in System V). These bits should only be
 22418 required to be cleared on regular files that have one or more of their execute bits set.

22419 The purpose of the *fchownat()* function is to enable changing ownership of files in directories
 22420 other than the current working directory without exposure to race conditions. Any part of the
 22421 path of a file could be changed in parallel to a call to *chown()* or *lchown()*, resulting in
 22422 unspecified behavior. By opening a file descriptor for the target directory and using the
 22423 *fchownat()* function it can be guaranteed that the changed file is located relative to the desired
 22424 directory.

22425 FUTURE DIRECTIONS

22426 None.

22427 SEE ALSO

22428 *chmod()*, *fpathconf()*, *lchown()*

22429 XBD <fcntl.h>, <sys/types.h>, <unistd.h>

22430 CHANGE HISTORY

22431 First released in Issue 1. Derived from Issue 1 of the SVID.

22432 Issue 6

22433 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 22434 • The wording describing the optional dependency on _POSIX_CHOWN_RESTRICTED is
- 22435 restored.

chown()*System Interfaces*

- 22436 • The [EPERM] error is restored as an error dependent on _POSIX_CHOWN_RESTRICTED.
 22437 This is since its operand is a pathname and applications should be aware that the error
 22438 may not occur for that pathname if the file system does not support
 22439 _POSIX_CHOWN_RESTRICTED.
- 22440 The following new requirements on POSIX implementations derive from alignment with the
 22441 Single UNIX Specification:
- 22442 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
 22443 required for conforming implementations of previous POSIX specifications, it was not
 22444 required for UNIX applications.
- 22445 • The value for *owner* of (**uid_t**)–1 allows the use of –1 by the owner of a file to change the
 22446 group ID only. A corresponding change is made for group.
- 22447 • The [ELOOP] mandatory error condition is added.
- 22448 • The [EIO] and [EINTR] optional error conditions are added.
- 22449 • A second [ENAMETOOLONG] is added as an optional error condition.
- 22450 The following changes were made to align with the IEEE P1003.1a draft standard:
- 22451 • Clarification is added that the S_ISUID and S_ISGID bits do not need to be cleared when
 22452 the process has appropriate privileges.
- 22453 • The [ELOOP] optional error condition is added.
- 22454 **Issue 7**
- 22455 Austin Group Interpretation 1003.1-2001 #143 is applied.
- 22456 The *fchownat()* function is added from The Open Group Technical Standard, 2006, Extended API
 22457 Set Part 2.
- 22458 Changes are made related to support for finegrained timestamps.
- 22459 Changes are made to allow a directory to be opened for searching.
- 22460 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
 22461 pathname exists but is not a directory or a symbolic link to a directory.

22462 **NAME**

22463 cimag, cimagf, cimagl — complex imaginary functions

22464 **SYNOPSIS**

22465 #include <complex.h>

22466 double cimag(double complex z);

22467 float cimagf(float complex z);

22468 long double cimagl(long double complex z);

22469 **DESCRIPTION**

22470 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 22471 conflict between the requirements described here and the ISO C standard is unintentional. This
 22472 volume of POSIX.1-2008 defers to the ISO C standard.

22473 These functions shall compute the imaginary part of *z*.22474 **RETURN VALUE**

22475 These functions shall return the imaginary part value (as a real).

22476 **ERRORS**

22477 No errors are defined.

22478 **EXAMPLES**

22479 None.

22480 **APPLICATION USAGE**22481 For a variable *z* of complex type:22482 *z* == creal(*z*) + cimag(*z*)*I22483 **RATIONALE**

22484 None.

22485 **FUTURE DIRECTIONS**

22486 None.

22487 **SEE ALSO**22488 *carg()*, *conj()*, *cproj()*, *creal()*

22489 XBD <complex.h>

22490 **CHANGE HISTORY**

22491 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

clearerr()*System Interfaces*22492 **NAME**

22493 clearerr — clear indicators on a stream

22494 **SYNOPSIS**

22495 #include <stdio.h>

22496 void clearerr(FILE *stream);

22497 **DESCRIPTION**

22498 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22499 conflict between the requirements described here and the ISO C standard is unintentional. This
22500 volume of POSIX.1-2008 defers to the ISO C standard.

22501 The *clearerr()* function shall clear the end-of-file and error indicators for the stream to which
22502 *stream* points.

22503 **RETURN VALUE**22504 The *clearerr()* function shall not return a value.22505 **ERRORS**

22506 No errors are defined.

22507 **EXAMPLES**

22508 None.

22509 **APPLICATION USAGE**

22510 None.

22511 **RATIONALE**

22512 None.

22513 **FUTURE DIRECTIONS**

22514 None.

22515 **SEE ALSO**

22516 XBD <stdio.h>

22517 **CHANGE HISTORY**

22518 First released in Issue 1. Derived from Issue 1 of the SVID.

22519 NAME

22520 clock — report CPU time used

22521 SYNOPSIS

22522 #include <time.h>

22523 clock_t clock(void);

22524 DESCRIPTION

22525 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22526 conflict between the requirements described here and the ISO C standard is unintentional. This
 22527 volume of POSIX.1-2008 defers to the ISO C standard.

22528 The *clock()* function shall return the implementation's best approximation to the processor time
 22529 used by the process since the beginning of an implementation-defined era related only to the
 22530 process invocation.

22531 RETURN VALUE

22532 To determine the time in seconds, the value returned by *clock()* should be divided by the value
 22533 XSI of the macro `CLOCKS_PER_SEC`. `CLOCKS_PER_SEC` is defined to be one million in `<time.h>`.
 22534 If the processor time used is not available or its value cannot be represented, the function shall
 22535 return the value `(clock_t)-1`.

22536 ERRORS

22537 No errors are defined.

22538 EXAMPLES

22539 None.

22540 APPLICATION USAGE

22541 In order to measure the time spent in a program, *clock()* should be called at the start of the
 22542 program and its return value subtracted from the value returned by subsequent calls. The value
 22543 returned by *clock()* is defined for compatibility across systems that have clocks with different
 22544 resolutions. The resolution on any particular system need not be to microsecond accuracy.

22545 The value returned by *clock()* may wrap around on some implementations. For example, on a
 22546 machine with 32-bit values for `clock_t`, it wraps after 2 147 seconds or 36 minutes.

22547 RATIONALE

22548 None.

22549 FUTURE DIRECTIONS

22550 None.

22551 SEE ALSO

22552 *asctime()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*

22553 XBD `<time.h>`

22554 CHANGE HISTORY

22555 First released in Issue 1. Derived from Issue 1 of the SVID.

clock_getcpuclockid()

System Interfaces

22556 **NAME**22557 clock_getcpuclockid — access a process CPU-time clock (**ADVANCED REALTIME**)22558 **SYNOPSIS**

```
22559 CPT    #include <time.h>
22560        int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
```

22561 **DESCRIPTION**

22562 The *clock_getcpuclockid()* function shall return the clock ID of the CPU-time clock of the process
 22563 specified by *pid*. If the process described by *pid* exists and the calling process has permission, the
 22564 clock ID of this clock shall be returned in *clock_id*.

22565 If *pid* is zero, the *clock_getcpuclockid()* function shall return the clock ID of the CPU-time clock of
 22566 the process making the call, in *clock_id*.

22567 The conditions under which one process has permission to obtain the CPU-time clock ID of
 22568 other processes are implementation-defined.

22569 **RETURN VALUE**

22570 Upon successful completion, *clock_getcpuclockid()* shall return zero; otherwise, an error number
 22571 shall be returned to indicate the error.

22572 **ERRORS**

22573 The *clock_getcpuclockid()* function shall fail if:

22574 [EPERM] The requesting process does not have permission to access the CPU-time clock
 22575 for the process.

22576 The *clock_getcpuclockid()* function may fail if:

22577 [ESRCH] No process can be found corresponding to the process specified by *pid*.

22578 **EXAMPLES**

22579 None.

22580 **APPLICATION USAGE**

22581 The *clock_getcpuclockid()* function is part of the Process CPU-Time Clocks option and need not be
 22582 provided on all implementations.

22583 **RATIONALE**

22584 None.

22585 **FUTURE DIRECTIONS**

22586 None.

22587 **SEE ALSO**

22588 *clock_getres()*, *timer_create()*

22589 XBD **<time.h>**

22590 **CHANGE HISTORY**

22591 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

22592 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

System Interfaces

clock_getres()22593 **NAME**22594 `clock_getres`, `clock_gettime`, `clock_settime` — clock and timer functions22595 **SYNOPSIS**

```

22596 CX    #include <time.h>
22597
22597     int clock_getres(clockid_t clock_id, struct timespec *res);
22598     int clock_gettime(clockid_t clock_id, struct timespec *tp);
22599     int clock_settime(clockid_t clock_id, const struct timespec *tp);

```

22600 **DESCRIPTION**

22601 The `clock_getres()` function shall return the resolution of any clock. Clock resolutions are
 22602 implementation-defined and cannot be set by a process. If the argument *res* is not NULL, the
 22603 resolution of the specified clock shall be stored in the location pointed to by *res*. If *res* is NULL,
 22604 the clock resolution is not returned. If the *time* argument of `clock_settime()` is not a multiple of *res*,
 22605 then the value is truncated to a multiple of *res*.

22606 The `clock_gettime()` function shall return the current value *tp* for the specified clock, *clock_id*.

22607 The `clock_settime()` function shall set the specified clock, *clock_id*, to the value specified by *tp*.
 22608 Time values that are between two consecutive non-negative integer multiples of the resolution of
 22609 the specified clock shall be truncated down to the smaller multiple of the resolution.

22610 A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that
 22611 is meaningful only within a process). All implementations shall support a *clock_id* of
 22612 `CLOCK_REALTIME` as defined in `<time.h>`. This clock represents the clock measuring real time
 22613 for the system. For this clock, the values returned by `clock_gettime()` and specified by
 22614 `clock_settime()` represent the amount of time (in seconds and nanoseconds) since the Epoch. An
 22615 implementation may also support additional clocks. The interpretation of time values for these
 22616 clocks is unspecified.

22617 If the value of the `CLOCK_REALTIME` clock is set via `clock_settime()`, the new value of the clock
 22618 shall be used to determine the time of expiration for absolute time services based upon the
 22619 `CLOCK_REALTIME` clock. This applies to the time at which armed absolute timers expire. If the
 22620 absolute time requested at the invocation of such a time service is before the new value of the
 22621 clock, the time service shall expire immediately as if the clock had reached the requested time
 22622 normally.

22623 Setting the value of the `CLOCK_REALTIME` clock via `clock_settime()` shall have no effect on
 22624 threads that are blocked waiting for a relative time service based upon this clock, including the
 22625 `nanosleep()` function; nor on the expiration of relative timers based upon this clock.
 22626 Consequently, these time services shall expire when the requested relative interval elapses,
 22627 independently of the new or old value of the clock.

22628 MON If the Monotonic Clock option is supported, all implementations shall support a *clock_id* of
 22629 `CLOCK_MONOTONIC` defined in `<time.h>`. This clock represents the monotonic clock for the
 22630 system. For this clock, the value returned by `clock_gettime()` represents the amount of time (in
 22631 seconds and nanoseconds) since an unspecified point in the past (for example, system start-up
 22632 time, or the Epoch). This point does not change after system start-up time. The value of the
 22633 `CLOCK_MONOTONIC` clock cannot be set via `clock_settime()`. This function shall fail if it is
 22634 invoked with a *clock_id* argument of `CLOCK_MONOTONIC`.

22635 The effect of setting a clock via `clock_settime()` on armed per-process timers associated with a
 22636 clock other than `CLOCK_REALTIME` is implementation-defined.

22637 If the value of the `CLOCK_REALTIME` clock is set via `clock_settime()`, the new value of the clock
 22638 shall be used to determine the time at which the system shall awaken a thread blocked on an

clock_getres()*System Interfaces*

22639		absolute <i>clock_nanosleep()</i> call based upon the <code>CLOCK_REALTIME</code> clock. If the absolute time
22640		requested at the invocation of such a time service is before the new value of the clock, the call
22641		shall return immediately as if the clock had reached the requested time normally.
22642		Setting the value of the <code>CLOCK_REALTIME</code> clock via <i>clock_settime()</i> shall have no effect on any
22643		thread that is blocked on a relative <i>clock_nanosleep()</i> call. Consequently, the call shall return
22644		when the requested relative interval elapses, independently of the new or old value of the clock.
22645		Appropriate privileges to set a particular clock are implementation-defined.
22646	CPT	If <code>_POSIX_CPUTIME</code> is defined, implementations shall support clock ID values obtained by
22647		invoking <i>clock_getcpuclockid()</i> , which represent the CPU-time clock of a given process.
22648		Implementations shall also support the special <code>clockid_t</code> value
22649		<code>CLOCK_PROCESS_CPUTIME_ID</code> , which represents the CPU-time clock of the calling process
22650		when invoking one of the <i>clock_*</i> () or <i>timer_*</i> () functions. For these clock IDs, the values
22651		returned by <i>clock_gettime()</i> and specified by <i>clock_settime()</i> represent the amount of execution
22652		time of the process associated with the clock. Changing the value of a CPU-time clock via
22653		<i>clock_settime()</i> shall have no effect on the behavior of the sporadic server scheduling policy (see
22654		Scheduling Policies , on page 501).
22655	TCT	If <code>_POSIX_THREAD_CPUTIME</code> is defined, implementations shall support clock ID values
22656		obtained by invoking <i>pthread_getcpuclockid()</i> , which represent the CPU-time clock of a given
22657		thread. Implementations shall also support the special <code>clockid_t</code> value
22658		<code>CLOCK_THREAD_CPUTIME_ID</code> , which represents the CPU-time clock of the calling thread
22659		when invoking one of the <i>clock_*</i> () or <i>timer_*</i> () functions. For these clock IDs, the values
22660		returned by <i>clock_gettime()</i> and specified by <i>clock_settime()</i> shall represent the amount of
22661		execution time of the thread associated with the clock. Changing the value of a CPU-time clock
22662		via <i>clock_settime()</i> shall have no effect on the behavior of the sporadic server scheduling policy
22663		(see Scheduling Policies , on page 501).
22664		RETURN VALUE
22665		A return value of 0 shall indicate that the call succeeded. A return value of –1 shall indicate that
22666		an error occurred, and <i>errno</i> shall be set to indicate the error.
22667		ERRORS
22668		The <i>clock_getres()</i> , <i>clock_gettime()</i> , and <i>clock_settime()</i> functions shall fail if:
22669		[EINVAL] The <i>clock_id</i> argument does not specify a known clock.
22670		The <i>clock_settime()</i> function shall fail if:
22671		[EINVAL] The <i>tp</i> argument to <i>clock_settime()</i> is outside the range for the given clock ID.
22672		[EINVAL] The <i>tp</i> argument specified a nanosecond value less than zero or greater than or
22673		equal to 1 000 million.
22674	MON	[EINVAL] The value of the <i>clock_id</i> argument is <code>CLOCK_MONOTONIC</code> .
22675		The <i>clock_settime()</i> function may fail if:
22676		[EPERM] The requesting process does not have appropriate privileges to set the
22677		specified clock.

EXAMPLES

None.

APPLICATION USAGE

Note that the absolute value of the monotonic clock is meaningless (because its origin is arbitrary), and thus there is no need to set it. Furthermore, realtime applications can rely on the fact that the value of this clock is never set and, therefore, that time intervals measured with this clock will not be affected by calls to *clock_settime()*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getcpuclockid(), *clock_nanosleep()*, *ctime()*, *mq_receive()*, *mq_send()*, *nanosleep()*, *pthread_mutex_timedlock()*, *sem_timedwait()*, *time()*, *timer_create()*, *timer_getoverrun()*

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

The APPLICATION USAGE section is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added of the effect of resetting the clock resolution.

CPU-time clocks and the *clock_getcpuclockid()* function are added for alignment with IEEE Std 1003.1d-1999.

The following changes are added for alignment with IEEE Std 1003.1j-2000:

- The DESCRIPTION is updated as follows:

— The value returned by *clock_gettime()* for CLOCK_MONOTONIC is specified.

— The *clock_settime()* function failing for CLOCK_MONOTONIC is specified.

— The effects of *clock_settime()* on the *clock_nanosleep()* function with respect to CLOCK_REALTIME are specified.

- An [EINVAL] error is added to the ERRORS section, indicating that *clock_settime()* fails for CLOCK_MONOTONIC.

- The APPLICATION USAGE section notes that the CLOCK_MONOTONIC clock need not and shall not be set by *clock_settime()* since the absolute value of the CLOCK_MONOTONIC clock is meaningless.

- The *clock_nanosleep()*, *mq_timedreceive()*, *mq_timedsend()*, *pthread_mutex_timedlock()*, *sem_timedwait()*, *timer_create()*, and *timer_settime()* functions are added to the SEE ALSO section.

clock_getres()*System Interfaces***Issue 7**

22717 22718 Functionality relating to the Clock Selection option is moved to the Base.

22719 22720 The *clock_getres()*, *clock_gettime()*, and *clock_settime()* functions are moved from the Timers option to the Base.

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

System Interfaces

clock_nanosleep()**NAME**

clock_nanosleep — high resolution sleep with specifiable clock

SYNOPSIS

```
#include <time.h>

int clock_nanosleep(clockid_t clock_id, int flags,
    const struct timespec *rqtp, struct timespec *rmtp);
```

DESCRIPTION

If the flag `TIMER_ABSTIME` is not set in the *flags* argument, the `clock_nanosleep()` function shall cause the current thread to be suspended from execution until either the time interval specified by the *rqtp* argument has elapsed, or a signal is delivered to the calling thread and its action is to invoke a signal-catching function, or the process is terminated. The clock used to measure the time shall be the clock specified by *clock_id*.

If the flag `TIMER_ABSTIME` is set in the *flags* argument, the `clock_nanosleep()` function shall cause the current thread to be suspended from execution until either the time value of the clock specified by *clock_id* reaches the absolute time specified by the *rqtp* argument, or a signal is delivered to the calling thread and its action is to invoke a signal-catching function, or the process is terminated. If, at the time of the call, the time value specified by *rqtp* is less than or equal to the time value of the specified clock, then `clock_nanosleep()` shall return immediately and the calling process shall not be suspended.

The suspension time caused by this function may be longer than requested because the argument value is rounded up to an integer multiple of the sleep resolution, or because of the scheduling of other activity by the system. But, except for the case of being interrupted by a signal, the suspension time for the relative `clock_nanosleep()` function (that is, with the `TIMER_ABSTIME` flag not set) shall not be less than the time interval specified by *rqtp*, as measured by the corresponding clock. The suspension for the absolute `clock_nanosleep()` function (that is, with the `TIMER_ABSTIME` flag set) shall be in effect at least until the value of the corresponding clock reaches the absolute time specified by *rqtp*, except for the case of being interrupted by a signal.

The use of the `clock_nanosleep()` function shall have no effect on the action or blockage of any signal.

The `clock_nanosleep()` function shall fail if the *clock_id* argument refers to the CPU-time clock of the calling thread. It is unspecified whether *clock_id* values of other CPU-time clocks are allowed.

RETURN VALUE

If the `clock_nanosleep()` function returns because the requested time has elapsed, its return value shall be zero.

If the `clock_nanosleep()` function returns because it has been interrupted by a signal, it shall return the corresponding error value. For the relative `clock_nanosleep()` function, if the *rmtp* argument is non-NULL, the **timespec** structure referenced by it shall be updated to contain the amount of time remaining in the interval (the requested time minus the time actually slept). If the *rmtp* argument is NULL, the remaining time is not returned. The absolute `clock_nanosleep()` function has no effect on the structure referenced by *rmtp*.

If `clock_nanosleep()` fails, it shall return the corresponding error value.

clock_nanosleep()*System Interfaces***ERRORS**

The *clock_nanosleep()* function shall fail if:

- [EINTR] The *clock_nanosleep()* function was interrupted by a signal.
- [EINVAL] The *rntp* argument specified a nanosecond value less than zero or greater than or equal to 1 000 million; or the *TIMER_ABSTIME* flag was specified in *flags* and the *rntp* argument is outside the range for the clock specified by *clock_id*; or the *clock_id* argument does not specify a known clock, or specifies the CPU-time clock of the calling thread.
- [ENOTSUP] The *clock_id* argument specifies a clock for which *clock_nanosleep()* is not supported, such as a CPU-time clock.

EXAMPLES

None.

APPLICATION USAGE

Calling *clock_nanosleep()* with the value *TIMER_ABSTIME* not set in the *flags* argument and with a *clock_id* of *CLOCK_REALTIME* is equivalent to calling *nanosleep()* with the same *rntp* and *rntp* arguments.

RATIONALE

The *nanosleep()* function specifies that the system-wide clock *CLOCK_REALTIME* is used to measure the elapsed time for this time service. However, with the introduction of the monotonic clock *CLOCK_MONOTONIC* a new relative sleep function is needed to allow an application to take advantage of the special characteristics of this clock.

There are many applications in which a process needs to be suspended and then activated multiple times in a periodic way; for example, to poll the status of a non-interrupting device or to refresh a display device. For these cases, it is known that precise periodic activation cannot be achieved with a relative *sleep()* or *nanosleep()* function call. Suppose, for example, a periodic process that is activated at time *T0*, executes for a while, and then wants to suspend itself until time *T0+T*, the period being *T*. If this process wants to use the *nanosleep()* function, it must first call *clock_gettime()* to get the current time, then calculate the difference between the current time and *T0+T* and, finally, call *nanosleep()* using the computed interval. However, the process could be preempted by a different process between the two function calls, and in this case the interval computed would be wrong; the process would wake up later than desired. This problem would not occur with the absolute *clock_nanosleep()* function, since only one function call would be necessary to suspend the process until the desired time. In other cases, however, a relative sleep is needed, and that is why both functionalities are required.

Although it is possible to implement periodic processes using the timers interface, this implementation would require the use of signals, and the reservation of some signal numbers. In this regard, the reasons for including an absolute version of the *clock_nanosleep()* function in POSIX.1-2008 are the same as for the inclusion of the relative *nanosleep()*.

It is also possible to implement precise periodic processes using *pthread_cond_timedwait()*, in which an absolute timeout is specified that takes effect if the condition variable involved is never signaled. However, the use of this interface is unnatural, and involves performing other operations on mutexes and condition variables that imply an unnecessary overhead. Furthermore, *pthread_cond_timedwait()* is not available in implementations that do not support threads.

Although the interface of the relative and absolute versions of the new high resolution sleep service is the same *clock_nanosleep()* function, the *rntp* argument is only used in the relative sleep. This argument is needed in the relative *clock_nanosleep()* function to reissue the function

22810 call if it is interrupted by a signal, but it is not needed in the absolute *clock_nanosleep()* function
22811 call; if the call is interrupted by a signal, the absolute *clock_nanosleep()* function can be invoked
22812 again with the same *rtp* argument used in the interrupted call.

22813 FUTURE DIRECTIONS

22814 None.

22815 SEE ALSO

22816 *clock_getres()*, *nanosleep()*, *pthread_cond_timedwait()*, *sleep()*

22817 XBD <time.h>

22818 CHANGE HISTORY

22819 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

22820 Issue 7

22821 The *clock_nanosleep()* function is moved from the Clock Selection option to the Base.

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

clock_settime()*System Interfaces*22822 **NAME**

22823 clock_settime — clock and timer functions

22824 **SYNOPSIS**22825 CX `#include <time.h>`22826 `int clock_settime(clockid_t clock_id, const struct timespec *tp);`22827 **DESCRIPTION**22828 Refer to *clock_getres()*.

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

22829 NAME

22830 `clog`, `clogf`, `clogl` — complex natural logarithm functions

22831 SYNOPSIS

22832 `#include <complex.h>`

22833 `double complex clog(double complex z);`

22834 `float complex clogf(float complex z);`

22835 `long double complex clogl(long double complex z);`

22836 DESCRIPTION

22837 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 22838 conflict between the requirements described here and the ISO C standard is unintentional. This
 22839 volume of POSIX.1-2008 defers to the ISO C standard.

22840 These functions shall compute the complex natural (base e) logarithm of z , with a branch cut
 22841 along the negative real axis.

22842 RETURN VALUE

22843 These functions shall return the complex natural logarithm value, in the range of a strip
 22844 mathematically unbounded along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary
 22845 axis.

22846 ERRORS

22847 No errors are defined.

22848 EXAMPLES

22849 None.

22850 APPLICATION USAGE

22851 None.

22852 RATIONALE

22853 None.

22854 FUTURE DIRECTIONS

22855 None.

22856 SEE ALSO

22857 [`cexp\(\)`](#)

22858 XBD [`<complex.h>`](#)

22859 CHANGE HISTORY

22860 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

close()*System Interfaces*22861 **NAME**

22862 close — close a file descriptor

22863 **SYNOPSIS**

22864 #include <unistd.h>

22865 int close(int *fildes*);22866 **DESCRIPTION**

22867 The *close()* function shall deallocate the file descriptor indicated by *fildes*. To deallocate means to
 22868 make the file descriptor available for return by subsequent calls to *open()* or other functions that
 22869 allocate file descriptors. All outstanding record locks owned by the process on the file associated
 22870 with the file descriptor shall be removed (that is, unlocked).

22871 If *close()* is interrupted by a signal that is to be caught, it shall return -1 with *errno* set to [EINTR]
 22872 and the state of *fildes* is unspecified. If an I/O error occurred while reading from or writing to
 22873 the file system during *close()*, it may return -1 with *errno* set to [EIO]; if this error is returned, the
 22874 state of *fildes* is unspecified.

22875 When all file descriptors associated with a pipe or FIFO special file are closed, any data
 22876 remaining in the pipe or FIFO shall be discarded.

22877 When all file descriptors associated with an open file description have been closed, the open file
 22878 description shall be freed.

22879 If the link count of the file is 0, when all file descriptors associated with the file are closed, the
 22880 space occupied by the file shall be freed and the file shall no longer be accessible.

22881 OB XSR If a STREAMS-based *fildes* is closed and the calling process was previously registered to receive
 22882 a SIGPOLL signal for events associated with that STREAM, the calling process shall be
 22883 unregistered for events associated with the STREAM. The last *close()* for a STREAM shall cause
 22884 the STREAM associated with *fildes* to be dismantled. If O_NONBLOCK is not set and there have
 22885 been no signals posted for the STREAM, and if there is data on the module's write queue, *close()*
 22886 shall wait for an unspecified time (for each module and driver) for any output to drain before
 22887 dismantling the STREAM. The time delay can be changed via an I_SETCLTIME *ioctl()* request. If
 22888 the O_NONBLOCK flag is set, or if there are any pending signals, *close()* shall not wait for
 22889 output to drain, and shall dismantle the STREAM immediately.

22890 If the implementation supports STREAMS-based pipes, and *fildes* is associated with one end of a
 22891 pipe, the last *close()* shall cause a hangup to occur on the other end of the pipe. In addition, if the
 22892 other end of the pipe has been named by *fattach()*, then the last *close()* shall force the named end
 22893 to be detached by *fdetach()*. If the named end has no open file descriptors associated with it and
 22894 gets detached, the STREAM associated with that end shall also be dismantled.

22895 XSI If *fildes* refers to the master side of a pseudo-terminal, and this is the last close, a SIGHUP signal
 22896 shall be sent to the controlling process, if any, for which the slave side of the pseudo-terminal is
 22897 the controlling terminal. It is unspecified whether closing the master side of the pseudo-terminal
 22898 flushes all queued input and output.

22899 OB XSR If *fildes* refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message
 22900 may be sent to the master.

22901 When there is an outstanding cancelable asynchronous I/O operation against *fildes* when *close()*
 22902 is called, that I/O operation may be canceled. An I/O operation that is not canceled completes
 22903 as if the *close()* operation had not yet occurred. All operations that are not canceled shall
 22904 complete as if the *close()* blocked until the operations completed. The *close()* operation itself
 22905 need not block awaiting such I/O completion. Whether any I/O operation is canceled, and
 22906 which I/O operation may be canceled upon *close()*, is implementation-defined.

22907 SHM If a memory mapped file or a shared memory object remains referenced at the last close (that is,
 22908 a process has it mapped), then the entire contents of the memory object shall persist until the
 22909 SHM memory object becomes unreferenced. If this is the last close of a memory mapped file or a
 22910 shared memory object and the close results in the memory object becoming unreferenced, and
 22911 the memory object has been unlinked, then the memory object shall be removed.

22912 If *fd* refers to a socket, *close()* shall cause the socket to be destroyed. If the socket is in
 22913 connection-mode, and the *SO_LINGER* option is set for the socket with non-zero linger time,
 22914 and the socket has untransmitted data, then *close()* shall block for up to the current linger
 22915 interval until all data is transmitted.

22916 RETURN VALUE

22917 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 22918 indicate the error.

22919 ERRORS

22920 The *close()* function shall fail if:

22921 [EBADF] The *fd* argument is not a valid file descriptor.

22922 [EINTR] The *close()* function was interrupted by a signal.

22923 The *close()* function may fail if:

22924 [EIO] An I/O error occurred while reading from or writing to the file system.

22925 EXAMPLES

22926 Reassigning a File Descriptor

22927 The following example closes the file descriptor associated with standard output for the current
 22928 process, re-assigns standard output to a new file descriptor, and closes the original file descriptor
 22929 to clean up. This example assumes that the file descriptor 0 (which is the descriptor for standard
 22930 input) is not closed.

```
22931 #include <unistd.h>
22932 ...
22933 int pfd;
22934 ...
22935 close(1);
22936 dup(pfd);
22937 close(pfd);
22938 ...
```

22939 Incidentally, this is exactly what could be achieved using:

```
22940 dup2(pfd, 1);
22941 close(pfd);
```

close()*System Interfaces***Closing a File Descriptor**

In the following example, *close()* is used to close a file descriptor after an unsuccessful attempt is made to associate that file descriptor with a stream.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define LOCKFILE "/etc/ptmp"
...
int pfd;
FILE *fpfd;
...
if ((fpfd = fdopen (pfd, "w")) == NULL) {
    close(pfd);
    unlink(LOCKFILE);
    exit(1);
}
...
```

APPLICATION USAGE

An application that had used the *stdio* routine *fopen()* to open a file should use the corresponding *fclose()* routine rather than *close()*. Once a file is closed, the file descriptor no longer exists, since the integer corresponding to it no longer refers to a file.

RATIONALE

The use of interruptible device close routines should be discouraged to avoid problems with the implicit closes of file descriptors by *exec* and *exit()*. This volume of POSIX.1-2008 only intends to permit such behavior by specifying the [EINTR] error condition.

Note that the requirement for *close()* on a socket to block for up to the current linger interval is not conditional on the O_NONBLOCK setting.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6 (on page 494), *exec*, *fattach()*, *fclose()*, *fdetach()*, *fopen()*, *ioctl()*, *open()*, *unlink()*

XBD <unistd.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Issue 6

The DESCRIPTION related to a STREAMS-based file or pseudo-terminal is marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] error condition is added as an optional error.

- 22984 • The DESCRIPTION is updated to describe the state of the *fildev* file descriptor as
22985 unspecified if an I/O error occurs and an [EIO] error condition is returned.
- 22986 Text referring to sockets is added to the DESCRIPTION.
- 22987 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
22988 shared memory objects and memory mapped files (and not typed memory objects) are the types
22989 of memory objects to which the paragraph on last closes applies.
- 22990 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/12 is applied, correcting the XSH shaded
22991 text relating to the master side of a pseudo-terminal. The reason for the change is that the
22992 behavior of pseudo-terminals and regular terminals should be as much alike as possible in this
22993 case; the change achieves that and matches historical behavior.
- 22994 **Issue 7**
- 22995 Functionality relating to the XSI STREAMS option is marked obsolescent.
- 22996 Functionality relating to the Asynchronous Input and Output and Memory Mapped Files
22997 options is moved to the Base.
- 22998 Austin Group Interpretation 1003.1-2001 #139 is applied, clarifying that the requirement for
22999 *close()* on a socket to block for up to the current linger interval is not conditional on the
23000 O_NONBLOCK setting.

closedir()

System Interfaces

23001 **NAME**

23002 closedir — close a directory stream

23003 **SYNOPSIS**

23004 #include <dirent.h>

23005 int closedir(DIR *dirp);

23006 **DESCRIPTION**

23007 The *closedir()* function shall close the directory stream referred to by the argument *dirp*. Upon
 23008 return, the value of *dirp* may no longer point to an accessible object of the type **DIR**. If a file
 23009 descriptor is used to implement type **DIR**, that file descriptor shall be closed.

23010 **RETURN VALUE**

23011 Upon successful completion, *closedir()* shall return 0; otherwise, -1 shall be returned and *errno*
 23012 set to indicate the error.

23013 **ERRORS**23014 The *closedir()* function may fail if:23015 [EBADF] The *dirp* argument does not refer to an open directory stream.23016 [EINTR] The *closedir()* function was interrupted by a signal.23017 **EXAMPLES**23018 **Closing a Directory Stream**23019 The following program fragment demonstrates how the *closedir()* function is used.

```

23020       ...
23021       DIR *dir;
23022       struct dirent *dp;
23023       ...
23024       if ((dir = opendir (".")) == NULL) {
23025       ...
23026       }
23027       while ((dp = readdir (dir)) != NULL) {
23028       ...
23029       }
23030       closedir(dir);
23031       ...

```

23032 **APPLICATION USAGE**

23033 None.

23034 **RATIONALE**

23035 None.

23036 **FUTURE DIRECTIONS**

23037 None.

23038 **SEE ALSO**23039 *dirfd()*, *fdopendir()*

23040 XBD <dirent.h>

CHANGE HISTORY

First released in Issue 2.

Issue 6

In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EINTR] error condition is added as an optional error condition.

closelog()*System Interfaces*23051 **NAME**

23052 closelog, openlog, setlogmask, syslog — control system log

23053 **SYNOPSIS**

```

23054 XSI #include <syslog.h>
23055
23055 void closelog(void);
23056 void openlog(const char *ident, int logopt, int facility);
23057 int setlogmask(int maskpri);
23058 void syslog(int priority, const char *message, ... /* arguments */);

```

23059 **DESCRIPTION**

23060 The *syslog()* function shall send a message to an implementation-defined logging facility, which
 23061 may log it in an implementation-defined system log, write it to the system console, forward it to
 23062 a list of users, or forward it to the logging facility on another host over the network. The logged
 23063 message shall include a message header and a message body. The message header contains at
 23064 least a timestamp and a tag string.

23065 The message body is generated from the *message* and following arguments in the same manner
 23066 as if these were arguments to *printf()*, except that the additional conversion specification *%m*
 23067 shall be recognized; it shall convert no arguments, shall cause the output of the error message
 23068 string associated with the value of *errno* on entry to *syslog()*, and may be mixed with argument
 23069 specifications of the "*%n\$*" form. If a complete conversion specification with the *m* conversion
 23070 specifier character is not just *%m*, the behavior is undefined. A trailing <newline> may be added
 23071 if needed.

23072 Values of the *priority* argument are formed by OR'ing together a severity-level value and an
 23073 optional facility value. If no facility value is specified, the current default facility value is used.

23074 Possible values of severity level include:

23075	LOG_EMERG	A panic condition.
23076	LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system
23077		database.
23078	LOG_CRIT	Critical conditions, such as hard device errors.
23079	LOG_ERR	Errors.
23080	LOG_WARNING	
23081		Warning messages.
23082	LOG_NOTICE	Conditions that are not error conditions, but that may require special
23083		handling.
23084	LOG_INFO	Informational messages.
23085	LOG_DEBUG	Messages that contain information normally of use only when debugging a
23086		program.

23087 The facility indicates the application or system component generating the message. Possible
 23088 facility values include:

23089	LOG_USER	Messages generated by arbitrary processes. This is the default facility
23090		identifier if none is specified.
23091	LOG_LOCAL0	Reserved for local use.

System Interfaces

closelog()

23092	LOG_LOCAL1	Reserved for local use.
23093	LOG_LOCAL2	Reserved for local use.
23094	LOG_LOCAL3	Reserved for local use.
23095	LOG_LOCAL4	Reserved for local use.
23096	LOG_LOCAL5	Reserved for local use.
23097	LOG_LOCAL6	Reserved for local use.
23098	LOG_LOCAL7	Reserved for local use.
23099	The <i>openlog()</i> function shall set process attributes that affect subsequent calls to <i>syslog()</i> . The	
23100	<i>ident</i> argument is a string that is prepended to every message. The <i>logopt</i> argument indicates	
23101	logging options. Values for <i>logopt</i> are constructed by a bitwise-inclusive OR of zero or more of	
23102	the following:	
23103	LOG_PID	Log the process ID with each message. This is useful for identifying specific
23104		processes.
23105	LOG_CONS	Write messages to the system console if they cannot be sent to the logging
23106		facility. The <i>syslog()</i> function ensures that the process does not acquire the
23107		console as a controlling terminal in the process of writing the message.
23108	LOG_NDELAY	Open the connection to the logging facility immediately. Normally the open is
23109		delayed until the first message is logged. This is useful for programs that need
23110		to manage the order in which file descriptors are allocated.
23111	LOG_ODELAY	Delay open until <i>syslog()</i> is called.
23112	LOG_NOWAIT	Do not wait for child processes that may have been created during the course
23113		of logging the message. This option should be used by processes that enable
23114		notification of child termination using SIGCHLD, since <i>syslog()</i> may
23115		otherwise block waiting for a child whose exit status has already been
23116		collected.
23117	The <i>facility</i> argument encodes a default facility to be assigned to all messages that do not have an	
23118	explicit facility already encoded. The initial default facility is LOG_USER.	
23119	The <i>openlog()</i> and <i>syslog()</i> functions may allocate a file descriptor. It is not necessary to call	
23120	<i>openlog()</i> prior to calling <i>syslog()</i> .	
23121	The <i>closelog()</i> function shall close any open file descriptors allocated by previous calls to	
23122	<i>openlog()</i> or <i>syslog()</i> .	
23123	The <i>setlogmask()</i> function shall set the log priority mask for the current process to <i>maskpri</i> and	
23124	return the previous mask. If the <i>maskpri</i> argument is 0, the current log mask is not modified.	
23125	Calls by the current process to <i>syslog()</i> with a priority not set in <i>maskpri</i> shall be rejected. The	
23126	default log mask allows all priorities to be logged. A call to <i>openlog()</i> is not required prior to	
23127	calling <i>setlogmask()</i> .	
23128	Symbolic constants for use as values of the <i>logopt</i> , <i>facility</i> , <i>priority</i> , and <i>maskpri</i> arguments are	
23129	defined in the <syslog.h> header.	
23130	RETURN VALUE	
23131	The <i>setlogmask()</i> function shall return the previous log priority mask. The <i>closelog()</i> , <i>openlog()</i> ,	
23132	and <i>syslog()</i> functions shall not return a value.	

closelog()

System Interfaces

23133 **ERRORS**

23134 No errors are defined.

23135 **EXAMPLES**23136 **Using openlog()**

23137 The following example causes subsequent calls to *syslog()* to log the process ID with each
 23138 message, and to write messages to the system console if they cannot be sent to the logging
 23139 facility.

```
23140 #include <syslog.h>
23141 char *ident = "Process demo";
23142 int logopt = LOG_PID | LOG_CONS;
23143 int facility = LOG_USER;
23144 ...
23145 openlog(ident, logopt, facility);
```

23146 **Using setlogmask()**

23147 The following example causes subsequent calls to *syslog()* to accept error messages, and to reject
 23148 all other messages.

```
23149 #include <syslog.h>
23150 int result;
23151 int mask = LOG_MASK (LOG_ERR);
23152 ...
23153 result = setlogmask(mask);
```

23154 **Using syslog**

23155 The following example sends the message "This is a message" to the default logging
 23156 facility, marking the message as an error message generated by random processes.

```
23157 #include <syslog.h>
23158 char *message = "This is a message";
23159 int priority = LOG_ERR | LOG_USER;
23160 ...
23161 syslog(priority, message);
```

23162 **APPLICATION USAGE**

23163 None.

23164 **RATIONALE**

23165 None.

23166 **FUTURE DIRECTIONS**

23167 None.

23168 **SEE ALSO**23169 *fprintf()*23170 XBD *<syslog.h>*

CHANGE HISTORY

23171
23172 First released in Issue 4, Version 2.

Issue 5

23173
23174 Moved from X/OPEN UNIX extension to BASE.

Issue 6

23175
23176 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/13 is applied, correcting the EXAMPLES
23177 section.

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

confstr()*System Interfaces*23178 **NAME**23179 `confstr` — get configurable variables23180 **SYNOPSIS**23181 `#include <unistd.h>`23182 `size_t confstr(int name, char *buf, size_t len);`23183 **DESCRIPTION**

23184 The `confstr()` function shall return configuration-defined string values. Its use and purpose are
 23185 similar to `sysconf()`, but it is used where string values rather than numeric values are returned.

23186 The *name* argument represents the system variable to be queried. The implementation shall
 23187 support the following name values, defined in `<unistd.h>`. It may support others:

23188 `_CS_PATH`
 23189 `_CS_POSIX_V7_ILP32_OFF32_CFLAGS`
 23190 `_CS_POSIX_V7_ILP32_OFF32_LDFLAGS`
 23191 `_CS_POSIX_V7_ILP32_OFF32_LIBS`
 23192 `_CS_POSIX_V7_ILP32_OFFBIG_CFLAGS`
 23193 `_CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS`
 23194 `_CS_POSIX_V7_ILP32_OFFBIG_LIBS`
 23195 `_CS_POSIX_V7_LP64_OFF64_CFLAGS`
 23196 `_CS_POSIX_V7_LP64_OFF64_LDFLAGS`
 23197 `_CS_POSIX_V7_LP64_OFF64_LIBS`
 23198 `_CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS`
 23199 `_CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS`
 23200 `_CS_POSIX_V7_LPBIG_OFFBIG_LIBS`
 23201 `_CS_POSIX_V7_THREADS_CFLAGS`
 23202 `_CS_POSIX_V7_THREADS_LDFLAGS`
 23203 `_CS_POSIX_V7_WIDTH_RESTRICTED_ENVS`
 23204 `_CS_V7_ENV`
 23205 **OR** `_CS_POSIX_V6_ILP32_OFF32_CFLAGS`
 23206 `_CS_POSIX_V6_ILP32_OFF32_LDFLAGS`
 23207 `_CS_POSIX_V6_ILP32_OFF32_LIBS`
 23208 `_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS`
 23209 `_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS`
 23210 `_CS_POSIX_V6_ILP32_OFFBIG_LIBS`
 23211 `_CS_POSIX_V6_LP64_OFF64_CFLAGS`
 23212 `_CS_POSIX_V6_LP64_OFF64_LDFLAGS`
 23213 `_CS_POSIX_V6_LP64_OFF64_LIBS`
 23214 `_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS`
 23215 `_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS`
 23216 `_CS_POSIX_V6_LPBIG_OFFBIG_LIBS`
 23217 `_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS`
 23218 `_CS_V6_ENV`

23219 If *len* is not 0, and if *name* has a configuration-defined value, `confstr()` shall copy that value into
 23220 the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes,
 23221 including the terminating null, then `confstr()` shall truncate the string to *len*−1 bytes and null-
 23222 terminate the result. The application can detect that the string was truncated by comparing the
 23223 value returned by `confstr()` with *len*.

23224 If *len* is 0 and *buf* is a null pointer, then `confstr()` shall still return the integer value as defined
 23225 below, but shall not return a string. If *len* is 0 but *buf* is not a null pointer, the result is

23226 unspecified.

23227 After a call to:

23228 `confstr(_CS_V7_ENV, buf, sizeof(buf))`

23229 the string stored in *buf* will contain the <space>-separated list of variable=value environment
23230 variable pairs required by the implementation to create a conforming environment, as described
23231 in the implementations' conformance documentation.

23232 If the implementation supports the POSIX shell option, the string stored in *buf* after a call to:

23233 `confstr(_CS_PATH, buf, sizeof(buf))`

23234 can be used as a value of the *PATH* environment variable that accesses all of the standard
23235 utilities of POSIX.1-2008, if the return value is less than or equal to *sizeof(buf)*.

23236 RETURN VALUE

23237 If *name* has a configuration-defined value, *confstr()* shall return the size of buffer that would be
23238 needed to hold the entire configuration-defined value including the terminating null. If this
23239 return value is greater than *len*, the string returned in *buf* is truncated.

23240 If *name* is invalid, *confstr()* shall return 0 and set *errno* to indicate the error.

23241 If *name* does not have a configuration-defined value, *confstr()* shall return 0 and leave *errno*
23242 unchanged.

23243 ERRORS

23244 The *confstr()* function shall fail if:

23245 [EINVAL] The value of the *name* argument is invalid.

23246 EXAMPLES

23247 None.

23248 APPLICATION USAGE

23249 An application can distinguish between an invalid *name* parameter value and one that
23250 corresponds to a configurable variable that has no configuration-defined value by checking if
23251 *errno* is modified. This mirrors the behavior of *sysconf()*.

23252 The original need for this function was to provide a way of finding the configuration-defined
23253 default value for the environment variable *PATH*. Since *PATH* can be modified by the user to
23254 include directories that could contain utilities replacing the standard utilities in the Shell and
23255 Utilities volume of POSIX.1-2008, applications need a way to determine the system-supplied
23256 *PATH* environment variable value that contains the correct search path for the standard utilities.

23257 An application could use:

23258 `confstr(name, (char *)NULL, (size_t)0)`

23259 to find out how big a buffer is needed for the string value; use *malloc()* to allocate a buffer to
23260 hold the string; and call *confstr()* again to get the string. Alternately, it could allocate a fixed,
23261 static buffer that is big enough to hold most answers (perhaps 512 or 1 024 bytes), but then use
23262 *malloc()* to allocate a larger buffer if it finds that this is too small.

23263 RATIONALE

23264 Application developers can normally determine any configuration variable by means of reading
23265 from the stream opened by a call to:

23266 `popen("command -p getconf variable", "r");`

23267 The *confstr()* function with a *name* argument of *_CS_PATH* returns a string that can be used as a

confstr()*System Interfaces*

- 23268 *PATH* environment variable setting that will reference the standard shell and utilities as
23269 described in the Shell and Utilities volume of POSIX.1-2008.
- 23270 The *confstr()* function copies the returned string into a buffer supplied by the application instead
23271 of returning a pointer to a string. This allows a cleaner function in some implementations (such
23272 as those with lightweight threads) and resolves questions about when the application must copy
23273 the string returned.
- 23274 **FUTURE DIRECTIONS**
23275 None.
- 23276 **SEE ALSO**
23277 *exec*, *fpathconf()*, *sysconf()*
- 23278 XBD **<unistd.h>**
23279 XCU *c99*
- 23280 **CHANGE HISTORY**
23281 First released in Issue 4. Derived from the ISO POSIX-2 standard.
- 23282 **Issue 5**
23283 A table indicating the permissible values of *name* is added to the DESCRIPTION. All those
23284 marked EX are new in this version.
- 23285 **Issue 6**
23286 The Open Group Corrigendum U033/7 is applied. The return value for the case returning the
23287 size of the buffer now explicitly states that this includes the terminating null.
- 23288 The following new requirements on POSIX implementations derive from alignment with the
23289 Single UNIX Specification:
- 23290 • The DESCRIPTION is updated with new arguments which can be used to determine
 - 23291 configuration strings for C compiler flags, linker/loader flags, and libraries for each
 - 23292 different supported programming environment. This is a change to support data size
 - 23293 neutrality.
- 23294 The following changes were made to align with the IEEE P1003.1a draft standard:
- 23295 • The DESCRIPTION is updated to include text describing how *_CS_PATH* can be used to
 - 23296 obtain a *PATH* to access the standard utilities.
- 23297 The macros associated with the *c89* programming models are marked LEGACY and new
23298 equivalent macros associated with *c99* are introduced.
- 23299 **Issue 7**
23300 Austin Group Interpretation 1003.1-2001 #047 is applied, adding the *_CS_V7_ENV* variable.
23301 Austin Group Interpretations 1003.1-2001 #166 is applied to permit an additional compiler flag
23302 to enable threads.
- 23303 The V6 variables for the supported programming environments are marked obsolescent.
23304 The variables for the supported programming environments are updated to be V7.
23305 The LEGACY variables and obsolescent values are removed.

System Interfaces

conj()23306 **NAME**

23307 conj, conjf, conjl — complex conjugate functions

23308 **SYNOPSIS**

23309 #include <complex.h>

23310 double complex conj(double complex z);

23311 float complex conjf(float complex z);

23312 long double complex conjl(long double complex z);

23313 **DESCRIPTION**

23314 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23315 conflict between the requirements described here and the ISO C standard is unintentional. This
 23316 volume of POSIX.1-2008 defers to the ISO C standard.

23317 These functions shall compute the complex conjugate of *z*, by reversing the sign of its imaginary
 23318 part.

23319 **RETURN VALUE**

23320 These functions return the complex conjugate value.

23321 **ERRORS**

23322 No errors are defined.

23323 **EXAMPLES**

23324 None.

23325 **APPLICATION USAGE**

23326 None.

23327 **RATIONALE**

23328 None.

23329 **FUTURE DIRECTIONS**

23330 None.

23331 **SEE ALSO**23332 *carg()*, *cimag()*, *cproj()*, *creal()*

23333 XBD <complex.h>

23334 **CHANGE HISTORY**

23335 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

connect()*System Interfaces*23336 **NAME**23337 `connect` — connect a socket23338 **SYNOPSIS**23339 `#include <sys/socket.h>`

23340 `int connect(int socket, const struct sockaddr *address,`
 23341 `socklen_t address_len);`

23342 **DESCRIPTION**

23343 The `connect()` function shall attempt to make a connection on a connection-mode socket or to set
 23344 or reset the peer address of a connectionless-mode socket. The function takes the following
 23345 arguments:

23346	<i>socket</i>	Specifies the file descriptor associated with the socket.
23347	<i>address</i>	Points to a sockaddr structure containing the peer address. The length and 23348 format of the address depend on the address family of the socket.
23349	<i>address_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>address</i> 23350 argument.

23351 If the socket has not already been bound to a local address, `connect()` shall bind it to an address
 23352 which, unless the socket's address family is AF_UNIX, is an unused local address.

23353 If the initiating socket is not connection-mode, then `connect()` shall set the socket's peer address,
 23354 and no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all
 23355 datagrams are sent on subsequent `send()` functions, and limits the remote sender for subsequent
 23356 `recv()` functions. If the *sa_family* member of *address* is AF_UNSPEC, the socket's peer address
 23357 shall be reset. Note that despite no connection being made, the term "connected" is used to
 23358 describe a connectionless-mode socket for which a peer address has been set.

23359 If the initiating socket is connection-mode, then `connect()` shall attempt to establish a connection
 23360 to the address specified by the *address* argument. If the connection cannot be established
 23361 immediately and O_NONBLOCK is not set for the file descriptor for the socket, `connect()` shall
 23362 block for up to an unspecified timeout interval until the connection is established. If the timeout
 23363 interval expires before the connection is established, `connect()` shall fail and the connection
 23364 attempt shall be aborted. If `connect()` is interrupted by a signal that is caught while blocked
 23365 waiting to establish a connection, `connect()` shall fail and set *errno* to [EINTR], but the connection
 23366 request shall not be aborted, and the connection shall be established asynchronously.

23367 If the connection cannot be established immediately and O_NONBLOCK is set for the file
 23368 descriptor for the socket, `connect()` shall fail and set *errno* to [EINPROGRESS], but the connection
 23369 request shall not be aborted, and the connection shall be established asynchronously. Subsequent
 23370 calls to `connect()` for the same socket, before the connection is established, shall fail and set *errno*
 23371 to [EALREADY].

23372 When the connection has been established asynchronously, `pselect()`, `select()`, and `poll()` shall
 23373 indicate that the file descriptor for the socket is ready for writing.

23374 The socket in use may require the process to have appropriate privileges to use the `connect()`
 23375 function.

23376 **RETURN VALUE**

23377 Upon successful completion, `connect()` shall return 0; otherwise, -1 shall be returned and *errno*
 23378 set to indicate the error.

23379 ERRORS

- 23380 The *connect()* function shall fail if:
- 23381 [EADDRNOTAVAIL]
 23382 The specified address is not available from the local machine.
- 23383 [EAFNOSUPPORT]
 23384 The specified address is not a valid address for the address family of the
 23385 specified socket.
- 23386 [EALREADY] A connection request is already in progress for the specified socket.
- 23387 [EBADF] The *socket* argument is not a valid file descriptor.
- 23388 [ECONNREFUSED]
 23389 The target address was not listening for connections or refused the connection
 23390 request.
- 23391 [EINPROGRESS] O_NONBLOCK is set for the file descriptor for the socket and the connection
 23392 cannot be immediately established; the connection shall be established
 23393 asynchronously.
- 23394 [EINTR] The attempt to establish a connection was interrupted by delivery of a signal
 23395 that was caught; the connection shall be established asynchronously.
- 23396 [EISCONN] The specified socket is connection-mode and is already connected.
- 23397 [ENETUNREACH]
 23398 No route to the network is present.
- 23399 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 23400 [EPROTOTYPE] The specified address has a different type than the socket bound to the
 23401 specified peer address.
- 23402 [ETIMEDOUT] The attempt to connect timed out before a connection was made.
- 23403 If the address family of the socket is AF_UNIX, then *connect()* shall fail if:
- 23404 [EIO] An I/O error occurred while reading from or writing to the file system.
- 23405 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
 23406 in *address*.
- 23407 [ENAMETOOLONG]
 23408 The length of a component of a pathname is longer than {NAME_MAX}.
- 23409 [ENOENT] A component of the pathname does not name an existing file or the pathname
 23410 is an empty string.
- 23411 [ENOTDIR] A component of the path prefix of the pathname in *address* is not a directory, or
 23412 the pathname in *address* contains at least one non-*<slash>* character and ends
 23413 with one or more trailing *<slash>* characters and the last pathname
 23414 component names an existing file that is neither a directory nor a symbolic
 23415 link to a directory.
- 23416 The *connect()* function may fail if:
- 23417 [EACCES] Search permission is denied for a component of the path prefix; or write access
 23418 to the named socket is denied.

connect()*System Interfaces*

23419	[EADDRINUSE]	Attempt to establish a connection that uses addresses that are already in use.
23420	[ECONNRESET]	Remote host reset the connection request.
23421	[EHOSTUNREACH]	
23422		The destination host cannot be reached (probably because the host is down or
23423		a remote router cannot reach it).
23424	[EINVAL]	The <i>address_len</i> argument is not a valid length for the address family; or
23425		invalid address family in the sockaddr structure.
23426	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
23427		resolution of the pathname in <i>address</i> .
23428	[ENAMETOOLONG]	
23429		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
23430		symbolic link produced an intermediate result with a length that exceeds
23431		{PATH_MAX}.
23432	[ENETDOWN]	The local network interface used to reach the destination is down.
23433	[ENOBUFS]	No buffer space is available.
23434	[EOPNOTSUPP]	The socket is listening and cannot be connected.

EXAMPLES

23435 None.

APPLICATION USAGE

23438 If *connect()* fails, the state of the socket is unspecified. Conforming applications should close the

23439 file descriptor and create a new socket before attempting to reconnect.

RATIONALE

23440 None.

FUTURE DIRECTIONS

23442 None.

SEE ALSO

23445 *accept()*, *bind()*, *close()*, *getsockname()*, *poll()*, *pselect()*, *send()*, *shutdown()*, *socket()*

23446 XBD <sys/socket.h>

CHANGE HISTORY

23447 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

23449 The wording of the mandatory [ELOOP] error condition is updated, and a second optional

23450 [ELOOP] error condition is added.

Issue 7

23452 Austin Group Interpretation 1003.1-2001 #035 is applied, clarifying the description of connected

23453 sockets.

23454 Austin Group Interpretation 1003.1-2001 #143 is applied.

23455 Austin Group Interpretation 1003.1-2001 #188 is applied, changing the method used to reset a

23456 peer address for a datagram socket.

23457 SD5-XSH-ERN-185 is applied.

23458 **NAME**

23459 copysign, copysignf, copysignl — number manipulation function

23460 **SYNOPSIS**

23461 #include <math.h>

23462 double copysign(double *x*, double *y*);23463 float copysignf(float *x*, float *y*);23464 long double copysignl(long double *x*, long double *y*);23465 **DESCRIPTION**

23466 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23467 conflict between the requirements described here and the ISO C standard is unintentional. This
 23468 volume of POSIX.1-2008 defers to the ISO C standard.

23469 These functions shall produce a value with the magnitude of *x* and the sign of *y*. On
 23470 implementations that represent a signed zero but do not treat negative zero consistently in
 23471 arithmetic operations, these functions regard the sign of zero as positive.

23472 **RETURN VALUE**

23473 Upon successful completion, these functions shall return a value with the magnitude of *x* and
 23474 the sign of *y*.

23475 **ERRORS**

23476 No errors are defined.

23477 **EXAMPLES**

23478 None.

23479 **APPLICATION USAGE**

23480 None.

23481 **RATIONALE**

23482 None.

23483 **FUTURE DIRECTIONS**

23484 None.

23485 **SEE ALSO**23486 [signbit\(\)](#)23487 XBD [<math.h>](#)23488 **CHANGE HISTORY**

23489 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

cos()*System Interfaces*23490 **NAME**

23491 cos, cosf, cosl — cosine function

23492 **SYNOPSIS**

23493 #include <math.h>

23494 double cos(double x);

23495 float cosf(float x);

23496 long double cosl(long double x);

23497 **DESCRIPTION**

23498 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23499 conflict between the requirements described here and the ISO C standard is unintentional. This
 23500 volume of POSIX.1-2008 defers to the ISO C standard.

23501 These functions shall compute the cosine of their argument x , measured in radians.

23502 An application wishing to check for error situations should set *errno* to zero and call
 23503 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 23504 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 23505 zero, an error has occurred.

23506 **RETURN VALUE**23507 Upon successful completion, these functions shall return the cosine of x .

23508 MX If x is NaN, a NaN shall be returned.

23509 If x is ± 0 , the value 1.0 shall be returned.

23510 If x is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 23511 defined value shall be returned.

23512 **ERRORS**

23513 These functions shall fail if:

23514 MX **Domain Error** The x argument is $\pm \text{Inf}$.

23515 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 23516 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 23517 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 23518 shall be raised.

23519 **EXAMPLES**23520 **Taking the Cosine of a 45-Degree Angle**

23521 #include <math.h>

23522 ...

23523 double radians = 45 * M_PI / 180;

23524 double result;

23525 ...

23526 result = cos(radians);

23527 **APPLICATION USAGE**

23528 These functions may lose accuracy when their argument is near an odd multiple of $\pi/2$ or is far
 23529 from 0.

23530 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 23531 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23532 **RATIONALE**

23533 None.

23534 **FUTURE DIRECTIONS**

23535 None.

23536 **SEE ALSO**23537 *acos()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sin()*, *tan()*23538 XBD Section 4.19 (on page 116), **<math.h>**23539 **CHANGE HISTORY**

23540 First released in Issue 1. Derived from Issue 1 of the SVID.

23541 **Issue 5**23542 The DESCRIPTION is updated to indicate how an application should check for an error. This
23543 text was previously published in the APPLICATION USAGE section.23544 **Issue 6**23545 The *cosf()* and *cosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.23546 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
23547 revised to align with the ISO/IEC 9899:1999 standard.23548 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
23549 marked.

cosh()

System Interfaces

23550 **NAME**

23551 cosh, coshf, coshl — hyperbolic cosine functions

23552 **SYNOPSIS**

23553 #include <math.h>

23554 double cosh(double x);

23555 float coshf(float x);

23556 long double coshl(long double x);

23557 **DESCRIPTION**

23558 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23559 conflict between the requirements described here and the ISO C standard is unintentional. This
 23560 volume of POSIX.1-2008 defers to the ISO C standard.

23561 These functions shall compute the hyperbolic cosine of their argument *x*.

23562 An application wishing to check for error situations should set *errno* to zero and call
 23563 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 23564 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 23565 zero, an error has occurred.

23566 **RETURN VALUE**23567 Upon successful completion, these functions shall return the hyperbolic cosine of *x*.

23568 If the correct value would cause overflow, a range error shall occur and *cosh()*, *coshf()*, and
 23569 *coshl()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 23570 respectively.

23571 MX If *x* is NaN, a NaN shall be returned.23572 If *x* is ± 0 , the value 1.0 shall be returned.23573 If *x* is $\pm \text{Inf}$, $\pm \text{Inf}$ shall be returned.23574 **ERRORS**

23575 These functions shall fail if:

23576 Range Error The result would cause an overflow.

23577 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 23578 then *errno* shall be set to [ERANGE]. If the integer expression
 23579 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 23580 floating-point exception shall be raised.

23581 **EXAMPLES**

23582 None.

23583 **APPLICATION USAGE**

23584 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 23585 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23586 For IEEE Std 754-1985 **double**, $710.5 < |x|$ implies that *cosh*(*x*) has overflowed.23587 **RATIONALE**

23588 None.

23589 **FUTURE DIRECTIONS**

23590 None.

23591 **SEE ALSO**23592 *acosh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sinh()*, *tanh()*23593 XBD Section 4.19 (on page 116), **<math.h>**23594 **CHANGE HISTORY**

23595 First released in Issue 1. Derived from Issue 1 of the SVID.

23596 **Issue 5**23597 The DESCRIPTION is updated to indicate how an application should check for an error. This
23598 text was previously published in the APPLICATION USAGE section.23599 **Issue 6**23600 The *coshf()* and *coshl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.23601 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
23602 revised to align with the ISO/IEC 9899:1999 standard.23603 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
23604 marked.

cosl()*System Interfaces*23605 **NAME**

23606 cosl — cosine function

23607 **SYNOPSIS**

23608 #include <math.h>

23609 long double cosl(long double x);

23610 **DESCRIPTION**23611 Refer to *cos()*.

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

NAME

cpow, cpowf, cpowl — complex power functions

SYNOPSIS

#include <complex.h>

double complex cpow(double complex *x*, double complex *y*);float complex cpowf(float complex *x*, float complex *y*);long double complex cpowl(long double complex *x*,long double complex *y*);**DESCRIPTION**

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall compute the complex power function x^y , with a branch cut for the first parameter along the negative real axis.

RETURN VALUE

These functions shall return the complex power function value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO[cabs\(\)](#), [csqrt\(\)](#)XBD [<complex.h>](#)**CHANGE HISTORY**

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

cproj()

System Interfaces

23643 **NAME**23644 `cproj`, `cprojf`, `cprojl` — complex projection functions23645 **SYNOPSIS**23646 `#include <complex.h>`23647 `double complex cproj(double complex z);`23648 `float complex cprojf(float complex z);`23649 `long double complex cprojl(long double complex z);`23650 **DESCRIPTION**

23651 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 23652 conflict between the requirements described here and the ISO C standard is unintentional. This
 23653 volume of POSIX.1-2008 defers to the ISO C standard.

23654 These functions shall compute a projection of *z* onto the Riemann sphere: *z* projects to *z*, except
 23655 that all complex infinities (even those with one infinite part and one NaN part) project to
 23656 positive infinity on the real axis. If *z* has an infinite part, then *cproj(z)* shall be equivalent to:

23657 `INFINITY + I * copysign(0.0, cimag(z))`23658 **RETURN VALUE**

23659 These functions shall return the value of the projection onto the Riemann sphere.

23660 **ERRORS**

23661 No errors are defined.

23662 **EXAMPLES**

23663 None.

23664 **APPLICATION USAGE**

23665 None.

23666 **RATIONALE**

23667 Two topologies are commonly used in complex mathematics: the complex plane with its
 23668 continuum of infinities, and the Riemann sphere with its single infinity. The complex plane is
 23669 better suited for transcendental functions, the Riemann sphere for algebraic functions. The
 23670 complex types with their multiplicity of infinities provide a useful (though imperfect) model for
 23671 the complex plane. The *cproj()* function helps model the Riemann sphere by mapping all
 23672 infinities to one, and should be used just before any operation, especially comparisons, that
 23673 might give spurious results for any of the other infinities. Note that a complex value with one
 23674 infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is
 23675 infinite, the complex value is infinite independent of the value of the other part. For the same
 23676 reason, *cabs()* returns an infinity if its argument has an infinite part and a NaN part.

23677 **FUTURE DIRECTIONS**

23678 None.

23679 **SEE ALSO**23680 *carg()*, *cimag()*, *conj()*, *creal()*23681 XBD `<complex.h>`23682 **CHANGE HISTORY**

23683 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23684 **NAME**

23685 creal, crealf, creall — complex real functions

23686 **SYNOPSIS**

23687 #include <complex.h>

23688 double creal(double complex z);

23689 float crealf(float complex z);

23690 long double creall(long double complex z);

23691 **DESCRIPTION**

23692 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23693 conflict between the requirements described here and the ISO C standard is unintentional. This
 23694 volume of POSIX.1-2008 defers to the ISO C standard.

23695 These functions shall compute the real part of z.

23696 **RETURN VALUE**

23697 These functions shall return the real part value.

23698 **ERRORS**

23699 No errors are defined.

23700 **EXAMPLES**

23701 None.

23702 **APPLICATION USAGE**23703 For a variable z of type **complex**:

23704 z == creal(z) + cimag(z)*I

23705 **RATIONALE**

23706 None.

23707 **FUTURE DIRECTIONS**

23708 None.

23709 **SEE ALSO**

23710 carg(), cimag(), conj(), cproj()

23711 XBD <complex.h>

23712 **CHANGE HISTORY**

23713 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

creat()

System Interfaces

23714 **NAME**23715 **creat** — create a new file or rewrite an existing one23716 **SYNOPSIS**23717 OH `#include <sys/stat.h>`23718 `#include <fcntl.h>`23719 `int creat(const char *path, mode_t mode);`23720 **DESCRIPTION**23721 The *creat()* function shall behave as if it is implemented as follows:23722 `int creat(const char *path, mode_t mode)`23723 `{`23724 `return open(path, O_WRONLY|O_CREAT|O_TRUNC, mode);`23725 `}`23726 **RETURN VALUE**23727 Refer to *open()*.23728 **ERRORS**23729 Refer to *open()*.23730 **EXAMPLES**23731 **Creating a File**

23732 The following example creates the file `/tmp/file` with read and write permissions for the file
 23733 owner and read permission for group and others. The resulting file descriptor is assigned to the
 23734 *fd* variable.

23735 `#include <fcntl.h>`23736 `...`23737 `int fd;`23738 `mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;`23739 `char *filename = "/tmp/file";`23740 `...`23741 `fd = creat(filename, mode);`23742 `...`23743 **APPLICATION USAGE**

23744 None.

23745 **RATIONALE**

23746 The *creat()* function is redundant. Its services are also provided by the *open()* function. It has
 23747 been included primarily for historical purposes since many existing applications depend on it. It
 23748 is best considered a part of the C binding rather than a function that should be provided in other
 23749 languages.

23750 **FUTURE DIRECTIONS**

23751 None.

23752 **SEE ALSO**23753 *mknod()*, *open()*23754 XBD `<fcntl.h>`, `<sys/stat.h>`, `<sys/types.h>`

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

Issue 7

SD5-XSH-ERN-186 is applied.

crypt()*System Interfaces*23766 **NAME**23767 **crypt** — string encoding function (**CRYPT**)23768 **SYNOPSIS**

```
23769 XSI      #include <unistd.h>
23770 char *crypt(const char *key, const char *salt);
```

23771 **DESCRIPTION**23772 The *crypt()* function is a string encoding function. The algorithm is implementation-defined.

23773 The *key* argument points to a string to be encoded. The *salt* argument shall be a string of at least
 23774 two bytes in length not including the null character chosen from the set:

```
23775 a b c d e f g h i j k l m n o p q r s t u v w x y z
23776 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
23777 0 1 2 3 4 5 6 7 8 9 . /
```

23778 The first two bytes of this string may be used to perturb the encoding algorithm.

23779 The return value of *crypt()* points to static data that is overwritten by each call.23780 The *crypt()* function need not be thread-safe.23781 **RETURN VALUE**

23782 Upon successful completion, *crypt()* shall return a pointer to the encoded string. The first two
 23783 bytes of the returned value shall be those of the *salt* argument. Otherwise, it shall return a null
 23784 pointer and set *errno* to indicate the error.

23785 **ERRORS**23786 The *crypt()* function shall fail if:

23787 [ENOSYS] The functionality is not supported on this implementation.

23788 **EXAMPLES**23789 **Encoding Passwords**

23790 The following example finds a user database entry matching a particular user name and changes
 23791 the current password to a new password. The *crypt()* function generates an encoded version of
 23792 each password. The first call to *crypt()* produces an encoded version of the old password; that
 23793 encoded password is then compared to the password stored in the user database. The second
 23794 call to *crypt()* encodes the new password before it is stored.

23795 The *putpwent()* function, used in the following example, is not part of POSIX.1-2008.

```
23796 #include <unistd.h>
23797 #include <pwd.h>
23798 #include <string.h>
23799 #include <stdio.h>
23800 ...
23801 int valid_change;
23802 int pfd; /* Integer for file descriptor returned by open(). */
23803 FILE *fpfd; /* File pointer for use in putpwent(). */
23804 struct passwd *p;
23805 char user[100];
23806 char oldpasswd[100];
23807 char newpasswd[100];
23808 char savepasswd[100];
```

System Interfaces

crypt()

```

23809     ...
23810     valid_change = 0;
23811     while ((p = getpwent()) != NULL) {
23812         /* Change entry if found. */
23813         if (strcmp(p->pw_name, user) == 0) {
23814             if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
23815                 strcpy(savepasswd, crypt(newpasswd, user));
23816                 p->pw_passwd = savepasswd;
23817                 valid_change = 1;
23818             }
23819             else {
23820                 fprintf(stderr, "Old password is not valid\n");
23821             }
23822         }
23823         /* Put passwd entry into ptmp. */
23824         putpwent(p, fpfd);
23825     }

```

APPLICATION USAGE

The values returned by this function need not be portable among XSI-conformant systems.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

encrypt(), *setkey()*

XBD <unistd.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XSH-ERN-178 is applied.

csin()*System Interfaces*23843 **NAME**

23844 csin, csinf, csinl — complex sine functions

23845 **SYNOPSIS**

23846 #include <complex.h>

23847 double complex csin(double complex z);

23848 float complex csinf(float complex z);

23849 long double complex csinl(long double complex z);

23850 **DESCRIPTION**

23851 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23852 conflict between the requirements described here and the ISO C standard is unintentional. This
 23853 volume of POSIX.1-2008 defers to the ISO C standard.

23854 These functions shall compute the complex sine of z.

23855 **RETURN VALUE**

23856 These functions shall return the complex sine value.

23857 **ERRORS**

23858 No errors are defined.

23859 **EXAMPLES**

23860 None.

23861 **APPLICATION USAGE**

23862 None.

23863 **RATIONALE**

23864 None.

23865 **FUTURE DIRECTIONS**

23866 None.

23867 **SEE ALSO**23868 [casin\(\)](#)23869 XBD [<complex.h>](#)23870 **CHANGE HISTORY**

23871 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

*System Interfaces***csinh()**23872 **NAME**

23873 csinh, csinhf, csinhl — complex hyperbolic sine functions

23874 **SYNOPSIS**

23875 #include <complex.h>

23876 double complex csinh(double complex z);

23877 float complex csinhf(float complex z);

23878 long double complex csinhl(long double complex z);

23879 **DESCRIPTION**

23880 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23881 conflict between the requirements described here and the ISO C standard is unintentional. This
 23882 volume of POSIX.1-2008 defers to the ISO C standard.

23883 These functions shall compute the complex hyperbolic sine of z.

23884 **RETURN VALUE**

23885 These functions shall return the complex hyperbolic sine value.

23886 **ERRORS**

23887 No errors are defined.

23888 **EXAMPLES**

23889 None.

23890 **APPLICATION USAGE**

23891 None.

23892 **RATIONALE**

23893 None.

23894 **FUTURE DIRECTIONS**

23895 None.

23896 **SEE ALSO**23897 `casinh()`

23898 XBD <complex.h>

23899 **CHANGE HISTORY**

23900 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

csinl()*System Interfaces*23901 **NAME**

23902 csinl — complex sine functions

23903 **SYNOPSIS**

23904 #include <complex.h>

23905 long double complex csinl(long double complex z);

23906 **DESCRIPTION**23907 Refer to *csin()*.

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

23908 **NAME**

23909 csqrt, csqrtf, csqrtl — complex square root functions

23910 **SYNOPSIS**

23911 #include <complex.h>

23912 double complex csqrt(double complex z);

23913 float complex csqrtf(float complex z);

23914 long double complex csqrtl(long double complex z);

23915 **DESCRIPTION**

23916 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23917 conflict between the requirements described here and the ISO C standard is unintentional. This
 23918 volume of POSIX.1-2008 defers to the ISO C standard.

23919 These functions shall compute the complex square root of *z*, with a branch cut along the negative
 23920 real axis.

23921 **RETURN VALUE**

23922 These functions shall return the complex square root value, in the range of the right half-plane
 23923 (including the imaginary axis).

23924 **ERRORS**

23925 No errors are defined.

23926 **EXAMPLES**

23927 None.

23928 **APPLICATION USAGE**

23929 None.

23930 **RATIONALE**

23931 None.

23932 **FUTURE DIRECTIONS**

23933 None.

23934 **SEE ALSO**23935 *cabs()*, *cpow()*

23936 XBD <complex.h>

23937 **CHANGE HISTORY**

23938 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

ctan()*System Interfaces*23939 **NAME**23940 `ctan, ctanf, ctanl` — complex tangent functions23941 **SYNOPSIS**23942 `#include <complex.h>`23943 `double complex ctan(double complex z);`23944 `float complex ctanf(float complex z);`23945 `long double complex ctanl(long double complex z);`23946 **DESCRIPTION**

23947 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23948 conflict between the requirements described here and the ISO C standard is unintentional. This
 23949 volume of POSIX.1-2008 defers to the ISO C standard.

23950 These functions shall compute the complex tangent of *z*.23951 **RETURN VALUE**

23952 These functions shall return the complex tangent value.

23953 **ERRORS**

23954 No errors are defined.

23955 **EXAMPLES**

23956 None.

23957 **APPLICATION USAGE**

23958 None.

23959 **RATIONALE**

23960 None.

23961 **FUTURE DIRECTIONS**

23962 None.

23963 **SEE ALSO**23964 [*catan\(\)*](#)23965 XBD [*<complex.h>*](#)23966 **CHANGE HISTORY**

23967 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23968 **NAME**

23969 ctanh, ctanhf, ctanhl — complex hyperbolic tangent functions

23970 **SYNOPSIS**

23971 #include <complex.h>

23972 double complex ctanh(double complex z);

23973 float complex ctanhf(float complex z);

23974 long double complex ctanhl(long double complex z);

23975 **DESCRIPTION**

23976 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23977 conflict between the requirements described here and the ISO C standard is unintentional. This
 23978 volume of POSIX.1-2008 defers to the ISO C standard.

23979 These functions shall compute the complex hyperbolic tangent of z.

23980 **RETURN VALUE**

23981 These functions shall return the complex hyperbolic tangent value.

23982 **ERRORS**

23983 No errors are defined.

23984 **EXAMPLES**

23985 None.

23986 **APPLICATION USAGE**

23987 None.

23988 **RATIONALE**

23989 None.

23990 **FUTURE DIRECTIONS**

23991 None.

23992 **SEE ALSO**

23993 catanh()

23994 XBD <complex.h>

23995 **CHANGE HISTORY**

23996 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

ctanl()*System Interfaces*23997 **NAME**

23998 ctanl — complex tangent functions

23999 **SYNOPSIS**

24000 #include <complex.h>

24001 long double complex ctanl(long double complex z);

24002 **DESCRIPTION**24003 Refer to *ctan()*.

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

System Interfaces

ctermid()24004 **NAME**24005 **ctermid** — generate a pathname for the controlling terminal24006 **SYNOPSIS**

```
24007 CX #include <stdio.h>
24008 char *ctermid(char *s);
```

24009 **DESCRIPTION**

24010 The *ctermid()* function shall generate a string that, when used as a pathname, refers to the
 24011 current controlling terminal for the current process. If *ctermid()* returns a pathname, access to the
 24012 file is not guaranteed.

24013 The *ctermid()* function need not be thread-safe if called with a NULL parameter.

24014 **RETURN VALUE**

24015 If *s* is a null pointer, the string shall be generated in an area that may be static (and therefore may
 24016 be overwritten by each call), the address of which shall be returned. Otherwise, *s* is assumed to
 24017 point to a character array of at least *L_ctermid* bytes; the string is placed in this array and the
 24018 value of *s* shall be returned. The symbolic constant *L_ctermid* is defined in *<stdio.h>*, and shall
 24019 have a value greater than 0.

24020 The *ctermid()* function shall return an empty string if the pathname that would refer to the
 24021 controlling terminal cannot be determined, or if the function is unsuccessful.

24022 **ERRORS**

24023 No errors are defined.

24024 **EXAMPLES**24025 **Determining the Controlling Terminal for the Current Process**

24026 The following example returns a pointer to a string that identifies the controlling terminal for the
 24027 current process. The pathname for the terminal is stored in the array pointed to by the *ptr*
 24028 argument, which has a size of *L_ctermid* bytes, as indicated by the *term* argument.

```
24029 #include <stdio.h>
24030 ...
24031 char term[L_ctermid];
24032 char *ptr;
24033 ptr = ctermid(term);
```

24034 **APPLICATION USAGE**

24035 The difference between *ctermid()* and *ttyname()* is that *ttyname()* must be handed a file
 24036 descriptor and return a path of the terminal associated with that file descriptor, while *ctermid()*
 24037 returns a string (such as *"/dev/tty"*) that refers to the current controlling terminal if used as a
 24038 pathname.

24039 **RATIONALE**

24040 *L_ctermid* must be defined appropriately for a given implementation and must be greater than
 24041 zero so that array declarations using it are accepted by the compiler. The value includes the
 24042 terminating null byte.

24043 Conforming applications that use multiple threads cannot call *ctermid()* with NULL as the
 24044 parameter. If *s* is not NULL, the *ctermid()* function generates a string that, when used as a
 24045 pathname, refers to the current controlling terminal for the current process. If *s* is NULL, the
 24046 return value of *ctermid()* is undefined.

ctermid()*System Interfaces*

24047 There is no additional burden on the programmer—changing to use a hypothetical thread-safe
24048 version of *ctermid()* along with allocating a buffer is more of a burden than merely allocating a
24049 buffer. Application code should not assume that the returned string is short, as some
24050 implementations have more than two pathname components before reaching a logical device
24051 name.

24052 FUTURE DIRECTIONS

24053 None.

24054 SEE ALSO

24055 *ttynname()*

24056 XBD *<stdio.h>*

24057 CHANGE HISTORY

24058 First released in Issue 1. Derived from Issue 1 of the SVID.

24059 Issue 5

24060 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

24061 Issue 6

24062 The normative text is updated to avoid use of the term “must” for application requirements.

24063 Issue 7

24064 Austin Group Interpretation 1003.1-2001 #148 is applied, updating the RATIONALE.

System Interfaces

ctime()24065 **NAME**24066 *ctime*, *ctime_r* — convert a time value to a date and time string24067 **SYNOPSIS**

```

24068 OB      #include <time.h>
24069          char *ctime(const time_t *clock);
24070 OB CX     char *ctime_r(const time_t *clock, char *buf);

```

24071 **DESCRIPTION**

24072 CX For *ctime()*: The functionality described on this reference page is aligned with the ISO C
 24073 standard. Any conflict between the requirements described here and the ISO C standard is
 24074 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

24075 The *ctime()* function shall convert the time pointed to by *clock*, representing time in seconds
 24076 since the Epoch, to local time in the form of a string. It shall be equivalent to:

```
24077      asctime(localtime(clock))
```

24078 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
 24079 objects: a broken-down time structure and an array of **char**. Execution of any of the functions
 24080 may overwrite the information returned in either of these objects by any of the other functions.

24081 The *ctime()* function need not be thread-safe.

24082 The *ctime_r()* function shall convert the calendar time pointed to by *clock* to local time in exactly
 24083 the same form as *ctime()* and put the string into the array pointed to by *buf* (which shall be at
 24084 least 26 bytes in size) and return *buf*.

24085 Unlike *ctime()*, the thread-safe version *ctime_r()* is not required to set *tzname*.

24086 **RETURN VALUE**

24087 The *ctime()* function shall return the pointer returned by *asctime()* with that broken-down time
 24088 as an argument.

24089 CX Upon successful completion, *ctime_r()* shall return a pointer to the string pointed to by *buf*.
 24090 When an error is encountered, a null pointer shall be returned.

24091 **ERRORS**

24092 No errors are defined.

24093 **EXAMPLES**

24094 None.

24095 **APPLICATION USAGE**

24096 These functions are included only for compatibility with older implementations. They have
 24097 undefined behavior if the resulting string would be too long, so the use of these functions
 24098 should be discouraged. On implementations that do not detect output string length overflow, it
 24099 is possible to overflow the output buffers in such a way as to cause applications to fail, or
 24100 possible system security violations. Also, these functions do not support localized date and time
 24101 formats. To avoid these problems, applications should use *strftime()* to generate strings from
 24102 broken-down times.

24103 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.

24104 The *ctime_r()* function is thread-safe and shall return values in a user-supplied buffer instead of
 24105 possibly using a static data area that may be overwritten by each call.

24106 Attempts to use *ctime()* or *ctime_r()* for times before the Epoch or for times beyond the year 9999
 24107 produce undefined results. Refer to *asctime()* (on page 590).

ctime()*System Interfaces***24108 RATIONALE**

24109 The standard developers decided to mark the *ctime()* and *ctime_r()* functions obsolescent even
 24110 though they are in the ISO C standard due to the possibility of buffer overflow. The ISO C
 24111 standard also provides the *strftime()* function which can be used to avoid these problems.

24112 FUTURE DIRECTIONS

24113 These functions may be removed in a future version.

24114 SEE ALSO

24115 *asctime(), clock(), difftime(), gmtime(), localtime(), mktime(), strftime(), strptime(), time(), utime()*

24116 XBD <time.h>

24117 CHANGE HISTORY

24118 First released in Issue 1. Derived from Issue 1 of the SVID.

24119 Issue 5

24120 Normative text previously in the APPLICATION USAGE section is moved to the
 24121 DESCRIPTION.

24122 The *ctime_r()* function is included for alignment with the POSIX Threads Extension.

24123 A note indicating that the *ctime()* function need not be reentrant is added to the DESCRIPTION.

24124 Issue 6

24125 Extensions beyond the ISO C standard are marked.

24126 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

24127 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
 24128 its avoidance of possibly using a static data area.

24129 Issue 7

24130 Austin Group Interpretation 1003.1-2001 #156 is applied.

24131 SD5-XSH-ERN-25 is applied, updating the APPLICATION USAGE.

24132 Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent.

24133 The *ctime_r()* function is moved from the Thread-Safe Functions option to the Base.

*System Interfaces***daylight**24134 **NAME**

24135 daylight — daylight savings time flag

24136 **SYNOPSIS**

```
24137 XSI #include <time.h>  
24138 extern int daylight;
```

24139 **DESCRIPTION**24140 Refer to *tzset()*.

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

dbm_clearerr()

System Interfaces

24141 **NAME**

24142 dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey,
 24143 dbm_open, dbm_store — database functions

24144 **SYNOPSIS**

```
24145 XSI      #include <ndbm.h>

24146      int dbm_clearerr(DBM *db);
24147      void dbm_close(DBM *db);
24148      int dbm_delete(DBM *db, datum key);
24149      int dbm_error(DBM *db);
24150      datum dbm_fetch(DBM *db, datum key);
24151      datum dbm_firstkey(DBM *db);
24152      datum dbm_nextkey(DBM *db);
24153      DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
24154      int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

24155 **DESCRIPTION**

24156 These functions create, access, and modify a database.

24157 A **datum** consists of at least two members, *dptr* and *dsize*. The *dptr* member points to an object
 24158 that is *dsize* bytes in length. Arbitrary binary data, as well as character strings, may be stored in
 24159 the object pointed to by *dptr*.

24160 A database shall be stored in one or two files. When one file is used, the name of the database
 24161 file shall be formed by appending the suffix **.db** to the *file* argument given to *dbm_open()*. When
 24162 two files are used, the names of the database files shall be formed by appending the suffixes **.dir**
 24163 and **.pag** respectively to the *file* argument.

24164 The *dbm_open()* function shall open a database. The *file* argument to the function is the
 24165 pathname of the database. The *open_flags* argument has the same meaning as the *flags* argument
 24166 of *open()* except that a database opened for write-only access opens the files for read and write
 24167 access and the behavior of the **O_APPEND** flag is unspecified. The *file_mode* argument has the
 24168 same meaning as the third argument of *open()*.

24169 The *dbm_open()* function need not accept pathnames longer than {PATH_MAX}–4 bytes
 24170 (including the terminating null), or pathnames with a last component longer than
 24171 {NAME_MAX}–4 bytes (excluding the terminating null).

24172 The *dbm_close()* function shall close a database. The application shall ensure that argument *db* is
 24173 a pointer to a **dbm** structure that has been returned from a call to *dbm_open()*.

24174 These database functions shall support an internal block size large enough to support
 24175 key/content pairs of at least 1 023 bytes.

24176 The *dbm_fetch()* function shall read a record from a database. The argument *db* is a pointer to a
 24177 database structure that has been returned from a call to *dbm_open()*. The argument *key* is a
 24178 **datum** that has been initialized by the application to the value of the key that matches the key of
 24179 the record the program is fetching.

24180 The *dbm_store()* function shall write a record to a database. The argument *db* is a pointer to a
 24181 database structure that has been returned from a call to *dbm_open()*. The argument *key* is a
 24182 **datum** that has been initialized by the application to the value of the key that identifies (for
 24183 subsequent reading, writing, or deleting) the record the application is writing. The argument
 24184 *content* is a **datum** that has been initialized by the application to the value of the record the
 24185 program is writing. The argument *store_mode* controls whether *dbm_store()* replaces any pre-
 24186 existing record that has the same key that is specified by the *key* argument. The application shall

set *store_mode* to either DBM_INSERT or DBM_REPLACE. If the database contains a record that matches the *key* argument and *store_mode* is DBM_REPLACE, the existing record shall be replaced with the new record. If the database contains a record that matches the *key* argument and *store_mode* is DBM_INSERT, the existing record shall be left unchanged and the new record ignored. If the database does not contain a record that matches the *key* argument and *store_mode* is either DBM_INSERT or DBM_REPLACE, the new record shall be inserted in the database.

If the sum of a key/content pair exceeds the internal block size, the result is unspecified. Moreover, the application shall ensure that all key/content pairs that hash together fit on a single block. The *dbm_store()* function shall return an error in the event that a disk block fills with inseparable data.

The *dbm_delete()* function shall delete a record and its key from the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open()*. The argument *key* is a **datum** that has been initialized by the application to the value of the key that identifies the record the program is deleting.

The *dbm_firstkey()* function shall return the first key in the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open()*.

The *dbm_nextkey()* function shall return the next key in the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open()*. The application shall ensure that the *dbm_firstkey()* function is called before calling *dbm_nextkey()*. Subsequent calls to *dbm_nextkey()* return the next key until all of the keys in the database have been returned.

The *dbm_error()* function shall return the error condition of the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open()*.

The *dbm_clearerr()* function shall clear the error condition of the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open()*.

The *dptr* pointers returned by these functions may point into static storage that may be changed by subsequent calls.

These functions need not be thread-safe.

RETURN VALUE

The *dbm_store()* and *dbm_delete()* functions shall return 0 when they succeed and a negative value when they fail.

The *dbm_store()* function shall return 1 if it is called with a *flags* value of DBM_INSERT and the function finds an existing record with the same key.

The *dbm_error()* function shall return 0 if the error condition is not set and return a non-zero value if the error condition is set.

The return value of *dbm_clearerr()* is unspecified.

The *dbm_firstkey()* and *dbm_nextkey()* functions shall return a key **datum**. When the end of the database is reached, the *dptr* member of the key is a null pointer. If an error is detected, the *dptr* member of the key shall be a null pointer and the error condition of the database shall be set.

The *dbm_fetch()* function shall return a content **datum**. If no record in the database matches the key or if an error condition has been detected in the database, the *dptr* member of the content shall be a null pointer.

The *dbm_open()* function shall return a pointer to a database structure. If an error is detected during the operation, *dbm_open()* shall return a (DBM *)0.

dbm_clearerr()*System Interfaces*24231 **ERRORS**

24232 No errors are defined.

24233 **EXAMPLES**

24234 None.

24235 **APPLICATION USAGE**

24236 The following code can be used to traverse the database:

24237

```
for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

24238 The *dbm_** functions provided in this library should not be confused in any way with those of a
 24239 general-purpose database management system. These functions do not provide for multiple
 24240 search keys per entry, they do not protect against multi-user access (in other words they do not
 24241 lock records or files), and they do not provide the many other useful database functions that are
 24242 found in more robust database management systems. Creating and updating databases by use of
 24243 these functions is relatively slow because of data copies that occur upon hash collisions. These
 24244 functions are useful for applications requiring fast lookup of relatively static information that is
 24245 to be indexed by a single key.

24246 Note that a strictly conforming application is extremely limited by these functions: since there is
 24247 no way to determine that the keys in use do not all hash to the same value (although that would
 24248 be rare), a strictly conforming application cannot be guaranteed that it can store more than one
 24249 block's worth of data in the database. As long as a key collision does not occur, additional data
 24250 may be stored, but because there is no way to determine whether an error is due to a key
 24251 collision or some other error condition (*dbm_error()* being effectively a Boolean), once an error is
 24252 detected, the application is effectively limited to guessing what the error might be if it wishes to
 24253 continue using these functions.

24254 The *dbm_delete()* function need not physically reclaim file space, although it does make it
 24255 available for reuse by the database.

24256 After calling *dbm_store()* or *dbm_delete()* during a pass through the keys by *dbm_firstkey()* and
 24257 *dbm_nextkey()*, the application should reset the database by calling *dbm_firstkey()* before again
 24258 calling *dbm_nextkey()*. The contents of these files are unspecified and may not be portable.

24259 Applications should take care that database pathname arguments specified to *dbm_open()* are
 24260 not prefixes of unrelated files. This might be done, for example, by placing databases in a
 24261 separate directory.

24262 Since some implementations use three characters for a suffix and others use four characters for a
 24263 suffix, applications should ensure that the maximum portable pathname length passed to
 24264 *dbm_open()* is no greater than {PATH_MAX}-4 bytes, with the last component of the pathname
 24265 no greater than {NAME_MAX}-4 bytes.

24266 **RATIONALE**

24267 Previously the standard required the database to be stored in two files, one file being a directory
 24268 containing a bitmap of keys and having **.dir** as its suffix. The second file containing all data and
 24269 having **.pag** as its suffix. This has been changed not to specify the use of the files and to allow
 24270 newer implementations of the Berkeley DB interface using a single file that have evolved while
 24271 remaining compatible with the application programming interface. The standard developers
 24272 considered removing the specific suffixes altogether but decided to retain them so as not to
 24273 pollute the application file name space more than necessary and to allow for portable backups of
 24274 the database.

24275 **FUTURE DIRECTIONS**

24276 None.

24277 **SEE ALSO**24278 *open()*24279 XBD *<ndbm.h>*24280 **CHANGE HISTORY**

24281 First released in Issue 4, Version 2.

24282 **Issue 5**

24283 Moved from X/OPEN UNIX extension to BASE.

24284 Normative text previously in the APPLICATION USAGE section is moved to the
24285 DESCRIPTION.

24286 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

24287 **Issue 6**

24288 The normative text is updated to avoid use of the term “must” for application requirements.

24289 **Issue 7**24290 Austin Group Interpretation 1003.1-2001 #042 is applied so that the DESCRIPTION permits
24291 newer implementations of the Berkeley DB interface.

24292 Austin Group Interpretation 1003.1-2001 #156 is applied.

difftime()

System Interfaces

24293 **NAME**24294 **difftime** — compute the difference between two calendar time values24295 **SYNOPSIS**24296 `#include <time.h>`24297 `double difftime(time_t time1, time_t time0);`24298 **DESCRIPTION**

24299 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 24300 conflict between the requirements described here and the ISO C standard is unintentional. This
 24301 volume of POSIX.1-2008 defers to the ISO C standard.

24302 The *difftime()* function shall compute the difference between two calendar times (as returned by
 24303 *time()*): *time1* − *time0*.

24304 **RETURN VALUE**24305 The *difftime()* function shall return the difference expressed in seconds as a type **double**.24306 **ERRORS**

24307 No errors are defined.

24308 **EXAMPLES**

24309 None.

24310 **APPLICATION USAGE**

24311 None.

24312 **RATIONALE**

24313 None.

24314 **FUTURE DIRECTIONS**

24315 None.

24316 **SEE ALSO**24317 *asctime()*, *clock()*, *ctime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*24318 XBD **<time.h>**24319 **CHANGE HISTORY**

24320 First released in Issue 4. Derived from the ISO C standard.

NAME

dirfd — extract the file descriptor used by a DIR stream

SYNOPSIS

```
#include <dirent.h>

int dirfd(DIR *dirp);
```

DESCRIPTION

The *dirfd()* function shall return a file descriptor referring to the same directory as the *dirp* argument. This file descriptor shall be closed by a call to *closedir()*. If any attempt is made to close the file descriptor, or to modify the state of the associated description, other than by means of *closedir()*, *readdir()*, *readdir_r()*, or *rewinddir()*, the behavior is undefined.

RETURN VALUE

Upon successful completion, the *dirfd()* function shall return an integer which contains a file descriptor for the stream pointed to by *dirp*. Otherwise, it shall return -1 and may set *errno* to indicate the error.

ERRORS

The *dirfd()* function may fail if:

[EINVAL]	The <i>dirp</i> argument does not refer to a valid directory stream.
[ENOTSUP]	The implementation does not support the association of a file descriptor with a directory.

EXAMPLES

None.

APPLICATION USAGE

The *dirfd()* function is intended to be a mechanism by which an application may obtain a file descriptor to use for the *fchdir()* function.

RATIONALE

This interface was introduced because the Base Definitions volume of POSIX.1-2008 does not make public the **DIR** data structure. Applications tend to use the *fchdir()* function on the file descriptor returned by this interface, and this has proven useful for security reasons; in particular, it is a better technique than others where directory names might change.

The description uses the term “a file descriptor” rather than “the file descriptor”. The implication intended is that an implementation that does not use an *fd* for *diropen()* could still *open()* the directory to implement the *dirfd()* function. Such a descriptor must be closed later during a call to *closedir()*.

An implementation that does not support file descriptors referring to directories may fail with [ENOTSUP].

If it is necessary to allocate an *fd* to be returned by *dirfd()*, it should be done at the time of a call to *opendir()*.

FUTURE DIRECTIONS

None.

SEE ALSO

closedir(), *fchdir()*, *fdopendir()*, *fileno()*, *open()*, *readdir()*

XBD **<dirent.h>**

dirfd()

System Interfaces

24363	CHANGE HISTORY
24364	First released in Issue 7.

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

System Interfaces

dirname()**NAME**

dirname — report the parent directory name of a file pathname

SYNOPSIS

```
#include <libgen.h>

char *dirname(char *path);
```

DESCRIPTION

The *dirname()* function shall take a pointer to a character string that contains a pathname, and return a pointer to a string that is a pathname of the parent directory of that file. Trailing '/' characters in the path are not counted as part of the path.

If *path* does not contain a '/', then *dirname()* shall return a pointer to the string ".". If *path* is a null pointer or points to an empty string, *dirname()* shall return a pointer to the string ".".

The *dirname()* function need not be thread-safe.

RETURN VALUE

The *dirname()* function shall return a pointer to a string that is the parent directory of *path*. If *path* is a null pointer or points to an empty string, a pointer to a string "." is returned.

The *dirname()* function may modify the string pointed to by *path*, and may return a pointer to static storage that may then be overwritten by subsequent calls to *dirname()*.

ERRORS

No errors are defined.

EXAMPLES

The following code fragment reads a pathname, changes the current working directory to the parent directory, and opens the file.

```
char *path = NULL, *pathcopy;
size_t buflen = 0;
ssize_t linelen = 0;
int fd;

linelen = getline(&path, &buflen, stdin);

path[linelen-1] = 0;
pathcopy = strdup(path);
if (chdir(dirname(pathcopy)) < 0) {
    ...
}
if ((fd = open(basename(path), O_RDONLY)) >= 0) {
    ...
    close (fd);
}
...
free (pathcopy);
free (path);
```

dirname()

24404 **Sample Input and Output Strings for `dirname()`**

24405 In the following table, the input string is the value pointed to by *path*, and the output string is
24406 the return value of the *dirname()* function.

24407		Input String	Output String
24408		<code>"/usr/lib"</code>	<code>"/usr"</code>
24409		<code>"/usr/"</code>	<code>"/"</code>
24410		<code>"usr"</code>	<code>."</code>
24411		<code>"/"</code>	<code>"/"</code>
24412		<code>."</code>	<code>."</code>
24413		<code> "."</code>	<code>."</code>

24414 **APPLICATION USAGE**

24415 The *dirname()* and *basename()* functions together yield a complete pathname. The expression
24416 *dirname(path)* obtains the pathname of the directory where *basename(path)* is found.

24417 Since the meaning of the leading `"/"` is implementation-defined, *dirname("//foo)* may return
24418 either `"/"` or `'/'` (but nothing else).

24419 **RATIONALE**

24420 None.

24421 **FUTURE DIRECTIONS**

24422 None.

24423 **SEE ALSO**

24424 *basename()*

24425 XBD `<libgen.h>`

24426 **CHANGE HISTORY**

24427 First released in Issue 4, Version 2.

24428 **Issue 5**

24429 Moved from X/OPEN UNIX extension to BASE.

24430 Normative text previously in the APPLICATION USAGE section is moved to the
24431 DESCRIPTION.

24432 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

24433 **Issue 7**

24434 Austin Group Interpretation 1003.1-2001 #156 is applied.

24435 The EXAMPLES section is revised.

System Interfaces

div()24436 **NAME**24437 `div` — compute the quotient and remainder of an integer division24438 **SYNOPSIS**24439 `#include <stdlib.h>`24440 `div_t div(int numer, int denom);`24441 **DESCRIPTION**

24442 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 24443 conflict between the requirements described here and the ISO C standard is unintentional. This
 24444 volume of POSIX.1-2008 defers to the ISO C standard.

24445 The `div()` function shall compute the quotient and remainder of the division of the numerator
 24446 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the integer
 24447 of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be
 24448 represented, the behavior is undefined; otherwise, $quot * denom + rem$ shall equal *numer*.

24449 **RETURN VALUE**

24450 The `div()` function shall return a structure of type `div_t`, comprising both the quotient and the
 24451 remainder. The structure includes the following members, in any order:

24452 `int quot; /* quotient */`
 24453 `int rem; /* remainder */`

24454 **ERRORS**

24455 No errors are defined.

24456 **EXAMPLES**

24457 None.

24458 **APPLICATION USAGE**

24459 None.

24460 **RATIONALE**

24461 None.

24462 **FUTURE DIRECTIONS**

24463 None.

24464 **SEE ALSO**24465 [*ldiv\(\)*](#)24466 XBD [`<stdlib.h>`](#)24467 **CHANGE HISTORY**

24468 First released in Issue 4. Derived from the ISO C standard.

dlclose()

System Interfaces

24469 **NAME**24470 **dlclose** — close a *dlopen()* object24471 **SYNOPSIS**24472 `#include <dlfcn.h>`24473 `int dlclosel(void *handle);`24474 **DESCRIPTION**

24475 The *dlclose()* function shall inform the system that the object referenced by a *handle* returned
 24476 from a previous *dlopen()* invocation is no longer needed by the application.

24477 The use of *dlclose()* reflects a statement of intent on the part of the process, but does not create
 24478 any requirement upon the implementation, such as removal of the code or symbols referenced
 24479 by *handle*. Once an object has been closed using *dlclose()* an application should assume that its
 24480 symbols are no longer available to *dlsym()*. All objects loaded automatically as a result of
 24481 invoking *dlopen()* on the referenced object shall also be closed if this is the last reference to it.

24482 Although a *dlclose()* operation is not required to remove structures from an address space,
 24483 neither is an implementation prohibited from doing so. The only restriction on such a removal is
 24484 that no object shall be removed to which references have been relocated, until or unless all such
 24485 references are removed. For instance, an object that had been loaded with a *dlopen()* operation
 24486 specifying the RTLD_GLOBAL flag might provide a target for dynamic relocations performed in
 24487 the processing of other objects—in such environments, an application may assume that no
 24488 relocation, once made, shall be undone or remade unless the object requiring the relocation has
 24489 itself been removed.

24490 **RETURN VALUE**

24491 If the referenced object was successfully closed, *dlclose()* shall return 0. If the object could not be
 24492 closed, or if *handle* does not refer to an open object, *dlclose()* shall return a non-zero value. More
 24493 detailed diagnostic information shall be available through *dLError()*.

24494 **ERRORS**

24495 No errors are defined.

24496 **EXAMPLES**24497 The following example illustrates use of *dlopen()* and *dlclose()*:

```

24498 ...
24499 /* Open a dynamic library and then close it ... */
25000 #include <dlfcn.h>
25001 void *mylib;
25002 int eret;
25003 mylib = dlopen("mylib.so", RTLD_LOCAL | RTLD_LAZY);
25004 ...
25005 eret = dlclosel(mylib);
25006 ...

```

24507 **APPLICATION USAGE**

24508 A conforming application should employ a *handle* returned from a *dlopen()* invocation only
 24509 within a given scope bracketed by the *dlopen()* and *dlclose()* operations. Implementations are
 24510 free to use reference counting or other techniques such that multiple calls to *dlopen()* referencing
 24511 the same object may return the same object for *handle*. Implementations are also free to reuse a
 24512 *handle*. For these reasons, the value of a *handle* must be treated as an opaque object by the
 24513 application, used only in calls to *dlsym()* and *dlclose()*.

24514 **RATIONALE**

24515 None.

24516 **FUTURE DIRECTIONS**

24517 None.

24518 **SEE ALSO**24519 *dlerror()*, *dlopen()*, *dlsym()*

24520 XBD <dlfcn.h>

24521 **CHANGE HISTORY**

24522 First released in Issue 5.

24523 **Issue 6**24524 The DESCRIPTION is updated to say that the referenced object is closed “if this is the last
24525 reference to it”.24526 **Issue 7**24527 The *dlopen()* function is moved from the XSI option to Base.

dlerror()*System Interfaces*24528 **NAME**24529 **dlerror** — get diagnostic information24530 **SYNOPSIS**24531 `#include <dlfcn.h>`24532 `char *dlerror(void);`24533 **DESCRIPTION**

24534 The *dlerror()* function shall return a null-terminated character string (with no trailing
 24535 <newline>) that describes the last error that occurred during dynamic linking processing. If no
 24536 dynamic linking errors have occurred since the last invocation of *dlerror()*, *dlerror()* shall return
 24537 NULL. Thus, invoking *dlerror()* a second time, immediately following a prior invocation, shall
 24538 result in NULL being returned.

24539 The *dlerror()* function need not be thread-safe.24540 **RETURN VALUE**

24541 If successful, *dlerror()* shall return a null-terminated character string; otherwise, NULL shall be
 24542 returned.

24543 **ERRORS**

24544 No errors are defined.

24545 **EXAMPLES**

24546 The following example prints out the last dynamic linking error:

```

24547 ...
24548 #include <dlfcn.h>
24549
24549 char *errstr;
24550
24550 errstr = dlerror();
24551 if (errstr != NULL)
24552     printf ("A dynamic linking error occurred: (%s)\n", errstr);
24553 ...

```

24554 **APPLICATION USAGE**

24555 The messages returned by *dlerror()* may reside in a static buffer that is overwritten on each call
 24556 to *dlerror()*. Application code should not write to this buffer. Programs wishing to preserve an
 24557 error message should make their own copies of that message. Depending on the application
 24558 environment with respect to asynchronous execution events, such as signals or other
 24559 asynchronous computation sharing the address space, conforming applications should use a
 24560 critical section to retrieve the error pointer and buffer.

24561 **RATIONALE**

24562 None.

24563 **FUTURE DIRECTIONS**

24564 None.

24565 **SEE ALSO**24566 *dlclose()*, *dlopen()*, *dlsym()*24567 XBD *<dlfcn.h>*24568 **CHANGE HISTORY**

24569 First released in Issue 5.

- 24570 **Issue 6**
24571 A note indicating that this function need not be reentrant is added to the DESCRIPTION.
- 24572 **Issue 7**
24573 Austin Group Interpretation 1003.1-2001 #156 is applied.
24574 The *dlerror()* function is moved from the XSI option to the Base.

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

dlopen()*System Interfaces*24575 **NAME**24576 **dlopen** — gain access to an executable object file24577 **SYNOPSIS**24578 `#include <dlfcn.h>`24579 `void *dlopen(const char *file, int mode);`24580 **DESCRIPTION**

24581 The *dlopen()* function shall make an executable object file specified by *file* available to the calling
 24582 program. The class of files eligible for this operation and the manner of their construction are
 24583 implementation-defined, though typically such files are executable objects such as shared
 24584 libraries, relocatable files, or programs. Note that some implementations permit the construction
 24585 of dependencies between such objects that are embedded within files. In such cases, a *dlopen()*
 24586 operation shall load such dependencies in addition to the object referenced by *file*.
 24587 Implementations may also impose specific constraints on the construction of programs that can
 24588 employ *dlopen()* and its related services.

24589 A successful *dlopen()* shall return a *handle* which the caller may use on subsequent calls to
 24590 *dlsym()* and *dlclose()*. The value of this *handle* should not be interpreted in any way by the caller.

24591 The *file* argument is used to construct a pathname to the object file. If *file* contains a <slash>
 24592 character, the *file* argument is used as the pathname for the file. Otherwise, *file* is used in an
 24593 implementation-defined manner to yield a pathname.

24594 If the value of *file* is 0, *dlopen()* shall provide a *handle* on a global symbol object. This object shall
 24595 provide access to the symbols from an ordered set of objects consisting of the original program
 24596 image file, together with any objects loaded at program start-up as specified by that process
 24597 image file (for example, shared libraries), and the set of objects loaded using a *dlopen()* operation
 24598 together with the RTLD_GLOBAL flag. As the latter set of objects can change during execution,
 24599 the set identified by *handle* can also change dynamically.

24600 Only a single copy of an object file is brought into the address space, even if *dlopen()* is invoked
 24601 multiple times in reference to the file, and even if different pathnames are used to reference the
 24602 file.

24603 The *mode* parameter describes how *dlopen()* shall operate upon *file* with respect to the processing
 24604 of relocations and the scope of visibility of the symbols provided within *file*. When an object is
 24605 brought into the address space of a process, it may contain references to symbols whose
 24606 addresses are not known until the object is loaded. These references shall be relocated before the
 24607 symbols can be accessed. The *mode* parameter governs when these relocations take place and
 24608 may have the following values:

24609 24610 24611 24612 24613 24614 24615 24616	RTLD_LAZY	Relocations shall be performed at an implementation-defined time, ranging from the time of the <i>dlopen()</i> call until the first reference to a given symbol occurs. Specifying RTLD_LAZY should improve performance on implementations supporting dynamic symbol binding as a process may not reference all of the functions in any given object. And, for systems supporting dynamic symbol resolution for normal process execution, this behavior mimics the normal handling of process execution.

24617 24618 24619 24620 24621	RTLD_NOW	All necessary relocations shall be performed when the object is first loaded. This may waste some processing if relocations are performed for functions that are never referenced. This behavior may be useful for applications that need to know as soon as an object is loaded that all symbols referenced during execution are available.

Any object loaded by *dlopen()* that requires relocations against global symbols can reference the symbols in the original process image file, any objects loaded at program start-up, from the object itself as well as any other object included in the same *dlopen()* invocation, and any objects that were loaded in any *dlopen()* invocation and which specified the RTLD_GLOBAL flag. To determine the scope of visibility for the symbols loaded with a *dlopen()* invocation, the *mode* parameter should be a bitwise-inclusive OR with one of the following values:

RTLD_GLOBAL The object's symbols shall be made available for the relocation processing of any other object. In addition, symbol lookup using *dlopen(0, mode)* and an associated *dlsym()* allows objects loaded with this *mode* to be searched.

RTLD_LOCAL The object's symbols shall not be made available for the relocation processing of any other object.

If neither RTLD_GLOBAL nor RTLD_LOCAL are specified, then the default behavior is unspecified.

If a *file* is specified in multiple *dlopen()* invocations, *mode* is interpreted at each invocation. Note, however, that once RTLD_NOW has been specified all relocations shall have been completed rendering further RTLD_NOW operations redundant and any further RTLD_LAZY operations irrelevant. Similarly, note that once RTLD_GLOBAL has been specified the object shall maintain the RTLD_GLOBAL status regardless of any previous or future specification of RTLD_LOCAL, as long as the object remains in the address space (see *dlclose()*).

Symbols introduced into a program through calls to *dlopen()* may be used in relocation activities. Symbols so introduced may duplicate symbols already defined by the program or previous *dlopen()* operations. To resolve the ambiguities such a situation might present, the resolution of a symbol reference to symbol definition is based on a symbol resolution order. Two such resolution orders are defined: *load* or *dependency* ordering. Load order establishes an ordering among symbol definitions, such that the definition first loaded (including definitions from the image file and any dependent objects loaded with it) has priority over objects added later (via *dlopen()*). Load ordering is used in relocation processing. Dependency ordering uses a breadth-first order starting with a given object, then all of its dependencies, then any dependents of those, iterating until all dependencies are satisfied. With the exception of the global symbol object obtained via a *dlopen()* operation on a *file* of 0, dependency ordering is used by the *dlsym()* function. Load ordering is used in *dlsym()* operations upon the global symbol object.

When an object is first made accessible via *dlopen()* it and its dependent objects are added in dependency order. Once all the objects are added, relocations are performed using load order. Note that if an object or its dependencies had been previously loaded, the load and dependency orders may yield different resolutions.

The symbols introduced by *dlopen()* operations and available through *dlsym()* are at a minimum those which are exported as symbols of global scope by the object. Typically such symbols shall be those that were specified in (for example) C source code as having *extern* linkage. The precise manner in which an implementation constructs the set of exported symbols for a *dlopen()* object is specified by that implementation.

RETURN VALUE

If *file* cannot be found, cannot be opened for reading, is not of an appropriate object format for processing by *dlopen()*, or if an error occurs during the process of loading *file* or relocating its symbolic references, *dlopen()* shall return NULL. More detailed diagnostic information shall be available through *dlerror()*.

dlopen()*System Interfaces*24667 **ERRORS**

24668 No errors are defined.

24669 **EXAMPLES**24670 Refer to *dlsym()* (on page 735).24671 **APPLICATION USAGE**

24672 None.

24673 **RATIONALE**

24674 None.

24675 **FUTURE DIRECTIONS**

24676 None.

24677 **SEE ALSO**24678 *dlclose()*, *dlderror()*, *dlsym()*24679 XBD **<dlfcn.h>**24680 **CHANGE HISTORY**

24681 First released in Issue 5.

24682 **Issue 6**

24683 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/21 is applied, changing the default
24684 behavior in the DESCRIPTION when neither RTLD_GLOBAL nor RTLD_LOCAL are specified
24685 from implementation-defined to unspecified.

24686 **Issue 7**24687 The *dlopen()* function is moved from the XSI option to the Base.24688 The EXAMPLES section is updated to refer to *dlsym()*.

NAME

dlsym — obtain the address of a symbol from a *dlopen()* object

SYNOPSIS

```
#include <dlfcn.h>
```

```
void *dlsym(void *restrict handle, const char *restrict name);
```

DESCRIPTION

The *dlsym()* function shall obtain the address of a symbol defined within an object made accessible through a *dlopen()* call. The *handle* argument is the value returned from a call to *dlopen()* (and which has not since been released via a call to *dlclose()*), and *name* is the symbol's name as a character string.

The *dlsym()* function shall search for the named symbol in all objects loaded automatically as a result of loading the object referenced by *handle* (see *dlopen()*). Load ordering is used in *dlsym()* operations upon the global symbol object. The symbol resolution algorithm used shall be dependency order as described in *dlopen()*.

The RTLD_DEFAULT and RTLD_NEXT flags are reserved for future use.

RETURN VALUE

If *handle* does not refer to a valid object opened by *dlopen()*, or if the named symbol cannot be found within any of the objects associated with *handle*, *dlsym()* shall return NULL. More detailed diagnostic information shall be available through *dlerror()*.

ERRORS

No errors are defined.

EXAMPLES

The following example shows how *dlopen()* and *dlsym()* can be used to access either function or data objects. For simplicity, error checking has been omitted.

```
void    *handle;
int      *iptr, (*fptr)(int);

/* open the needed object */
handle = dlopen("/usr/home/me/libfoo.so", RTLD_LOCAL | RTLD_LAZY);

/* find the address of function and data objects */
*(void **)(&fptr) = dlsym(handle, "my_function");
iptr = (int *)dlsym(handle, "my_object");

/* invoke function, passing value of integer as a parameter */
(*fptr)(*iptr);
```

APPLICATION USAGE

Special purpose values for *handle* are reserved for future use. These values and their meanings are:

RTLD_DEFAULT The symbol lookup happens in the normal global scope; that is, a search for a symbol using this handle would find the same definition as a direct use of this symbol in the program code.

RTLD_NEXT Specifies the next object after this one that defines *name*. *This one* refers to the object containing the invocation of *dlsym()*. The *next* object is the one found upon the application of a load order symbol resolution algorithm (see *dlopen()*). The next object is either one of global scope (because it was introduced as part of the original process image or because it was added with a *dlopen()* operation including the RTLD_GLOBAL flag), or is an object that

dlsym()*System Interfaces*

24734 was included in the same *dlopen()* operation that loaded this one.

24735 The RTLD_NEXT flag is useful to navigate an intentionally created hierarchy

24736 of multiply-defined symbols created through *interposition*. For example, if a

24737 program wished to create an implementation of *malloc()* that embedded some

24738 statistics gathering about memory allocations, such an implementation could

24739 use the real *malloc()* definition to perform the memory allocation—and itself

24740 only embed the necessary logic to implement the statistics gathering function.

RATIONALE

24741 The ISO C standard does not require that pointers to functions can be cast back and forth to

24742 pointers to data. However, POSIX-conforming implementations are required to support this, as

24743 noted in [Section 2.12.3](#) (on page 541). The result of converting a pointer to a function into a

24744 pointer to another data type (except **void ***) is still undefined, however.

24746 Note that compilers conforming to the ISO C standard are required to generate a warning if a

24747 conversion from a **void *** pointer to a function pointer is attempted as in:

24748 `fptr = (int (*)(int))dlsym(handle, "my_function");`

FUTURE DIRECTIONS

24749 None.

SEE ALSO

24752 [*dlclose\(\)*](#), [*dlerror\(\)*](#), [*dlopen\(\)*](#)

24753 XBD [*<dlfcn.h>*](#)

CHANGE HISTORY

24755 First released in Issue 5.

Issue 6

24756 The **restrict** keyword is added to the *dlsym()* prototype for alignment with the

24757 ISO/IEC 9899:1999 standard.

24759 The RTLD_DEFAULT and RTLD_NEXT flags are reserved for future use.

24760 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/14 is applied, correcting an example, and

24761 adding text to the RATIONALE describing issues related to conversion of pointers to functions

24762 and back again.

Issue 7

24763 The *dlsym()* function is moved from the XSI option to the Base.

*System Interfaces***dprintf()**24765 **NAME**

24766 dprintf — print formatted output

24767 **SYNOPSIS**

```
24768 CX       #include <stdio.h>  
24769       int dprintf(int fildes, const char *restrict format, ...);
```

24770 **DESCRIPTION**24771 Refer to *fprintf()*.

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

drand48()

System Interfaces

24772 **NAME**

24773 **drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48** — generate
 24774 uniformly distributed pseudo-random numbers

24775 **SYNOPSIS**

```
24776 XSI      #include <stdlib.h>

24777 double drand48(void);
24778 double erand48(unsigned short xsubi[3]);
24779 long jrand48(unsigned short xsubi[3]);
24780 void lcong48(unsigned short param[7]);
24781 long lrand48(void);
24782 long mrand48(void);
24783 long nrand48(unsigned short xsubi[3]);
24784 unsigned short *seed48(unsigned short seed16v[3]);
24785 void srand48(long seedval);
```

24786 **DESCRIPTION**

24787 This family of functions shall generate pseudo-random numbers using a linear congruential
 24788 algorithm and 48-bit integer arithmetic.

24789 The *drand48()* and *erand48()* functions shall return non-negative, double-precision, floating-
 24790 point values, uniformly distributed over the interval [0.0,1.0).

24791 The *lrand48()* and *nrand48()* functions shall return non-negative, long integers, uniformly
 24792 distributed over the interval [0,2³¹).

24793 The *mrnd48()* and *jrnd48()* functions shall return signed long integers uniformly distributed
 24794 over the interval [−2³¹,2³¹).

24795 The *srand48()*, *seed48()*, and *lcong48()* functions are initialization entry points, one of which
 24796 should be invoked before either *drand48()*, *lrand48()*, or *mrnd48()* is called. (Although it is not
 24797 recommended practice, constant default initializer values shall be supplied automatically if
 24798 *drand48()*, *lrand48()*, or *mrnd48()* is called without a prior call to an initialization entry point.)
 24799 The *erand48()*, *nrand48()*, and *jrnd48()* functions do not require an initialization entry point to
 24800 be called first.

24801 All the routines work by generating a sequence of 48-bit integer values, X_i , according to the
 24802 linear congruential formula:

$$24803 \quad X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n \geq 0$$

24804 The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless *lcong48()* is invoked,
 24805 the multiplier value a and the addend value c are given by:

$$24806 \quad a = 5\text{DEECE66D}_{16} = 273673163155_8$$

$$24807 \quad c = \text{B}_{16} = 13_8$$

24808 The value returned by any of the *drand48()*, *erand48()*, *jrnd48()*, *lrand48()*, *mrnd48()*, or
 24809 *nrand48()* functions is computed by first generating the next 48-bit X_i in the sequence. Then the
 24810 appropriate number of bits, according to the type of data item to be returned, are copied from
 24811 the high-order (leftmost) bits of X_i and transformed into the returned value.

24812 The *drand48()*, *lrand48()*, and *mrnd48()* functions store the last 48-bit X_i generated in an
 24813 internal buffer; that is why the application shall ensure that these are initialized prior to being
 24814 invoked. The *erand48()*, *nrand48()*, and *jrnd48()* functions require the calling program to

provide storage for the successive X_i values in the array specified as an argument when the functions are invoked. That is why these routines do not have to be initialized; the calling program merely has to place the desired initial value of X_i into the array and pass it as an argument. By using different arguments, *erand48()*, *rand48()*, and *jrand48()* allow separate modules of a large program to generate several *independent* streams of pseudo-random numbers; that is, the sequence of numbers in each stream shall *not* depend upon how many times the routines are called to generate numbers for the other streams.

The initializer function *srand48()* sets the high-order 32 bits of X_i to the low-order 32 bits contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

The initializer function *seed48()* sets the value of X_i to the 48-bit value specified in the argument array. The low-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[0]. The mid-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[1]. The high-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[2]. In addition, the previous value of X_i is copied into a 48-bit internal buffer, used only by *seed48()*, and a pointer to this buffer is the value returned by *seed48()*. This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time—use the pointer to get at and store the last X_i value, and then use this value to reinitialize via *seed48()* when the program is restarted.

The initializer function *lcong48()* allows the user to specify the initial X_i , the multiplier value a , and the addend value c . Argument array elements *param*[0-2] specify X_i , *param*[3-5] specify the multiplier a , and *param*[6] specifies the 16-bit addend c . After *lcong48()* is called, a subsequent call to either *srand48()* or *seed48()* shall restore the standard multiplier and addend values, a and c , specified above.

The *drand48()*, *lrnd48()*, and *mrnd48()* functions need not be thread-safe.

RETURN VALUE

As described in the DESCRIPTION above.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

rand()

XBD <stdlib.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A note indicating that the *drand48()*, *lrnd48()*, and *mrnd48()* functions need not be reentrant is added to the DESCRIPTION.

drand48()*System Interfaces*24858 **Issue 6**

24859 The normative text is updated to avoid use of the term “must” for application requirements.

24860 **Issue 7**

24861 Austin Group Interpretation 1003.1-2001 #156 is applied.

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

NAME

dup, dup2 — duplicate an open file descriptor

SYNOPSIS

```
#include <unistd.h>

int dup(int fildes);
int dup2(int fildes, int fildes2);
```

DESCRIPTION

The *dup()* function provides an alternative interface to the service provided by *fcntl()* using the *F_DUPFD* command. The call *dup(fildes)* shall be equivalent to:

```
fcntl(fildes, F_DUPFD, 0);
```

The *dup2()* function shall cause the file descriptor *fildes2* to refer to the same open file description as the file descriptor *fildes* and to share any locks, and shall return *fildes2*. If *fildes2* is already a valid open file descriptor, it shall be closed first, unless *fildes* is equal to *fildes2* in which case *dup2()* shall return *fildes2* without closing it. If the close operation fails to close *fildes2*, *dup2()* shall return *-1* without changing the open file description to which *fildes2* refers. If *fildes* is not a valid file descriptor, *dup2()* shall return *-1* and shall not close *fildes2*. If *fildes2* is less than 0 or greater than or equal to {*OPEN_MAX*}, *dup2()* shall return *-1* with *errno* set to [*EBADF*].

Upon successful completion, if *fildes* is not equal to *fildes2*, the *FD_CLOEXEC* flag associated with *fildes2* shall be cleared. If *fildes* is equal to *fildes2*, the *FD_CLOEXEC* flag associated with *fildes2* shall not be changed.

TYM If *fildes* refers to a typed memory object, the result of the *dup2()* function is unspecified.

RETURN VALUE

Upon successful completion a non-negative integer, namely the file descriptor, shall be returned; otherwise, *-1* shall be returned and *errno* set to indicate the error.

ERRORS

The *dup()* function shall fail if:

[*EBADF*] The *fildes* argument is not a valid open file descriptor.

[*EMFILE*] All file descriptors available to the process are currently open.

The *dup2()* function shall fail if:

[*EBADF*] The *fildes* argument is not a valid open file descriptor or the argument *fildes2* is negative or greater than or equal to {*OPEN_MAX*}.

[*EINTR*] The *dup2()* function was interrupted by a signal.

The *dup2()* function may fail if:

[*EIO*] An I/O error occurred while attempting to close *fildes2*.

EXAMPLES**Redirecting Standard Output to a File**

The following example closes standard output for the current processes, re-assigns standard output to go to the file referenced by *pfid*, and closes the original file descriptor to clean up.

```
#include <unistd.h>
...
int pfid;
...
```

dup()*System Interfaces*

```

24904     close(1);
24905     dup(pfd);
24906     close(pfd);
24907     ...

```

24908 **Redirecting Error Messages**

24909 The following example redirects messages from *stderr* to *stdout*.

```

24910     #include <unistd.h>
24911     ...
24912     dup2(1, 2);
24913     ...

```

24914 **APPLICATION USAGE**

24915 None.

24916 **RATIONALE**

24917 The *dup()* and *dup2()* functions are redundant. Their services are also provided by the *fcntl()*
 24918 function. They have been included in this volume of POSIX.1-2008 primarily for historical
 24919 reasons, since many existing applications use them.

24920 The *dup2()* function is not marked obsolescent because it presents a type-safe version of
 24921 functionality provided in a type-unsafe version by *fcntl()*. It is used in the POSIX Ada binding.

24922 The *dup2()* function is not intended for use in critical regions as a synchronization mechanism.

24923 In the description of [EBADF], the case of *fildev* being out of range is covered by the given case of
 24924 *fildev* not being valid. The descriptions for *fildev* and *fildev2* are different because the only kind of
 24925 invalidity that is relevant for *fildev2* is whether it is out of range; that is, it does not matter
 24926 whether *fildev2* refers to an open file when the *dup2()* call is made.

24927 **FUTURE DIRECTIONS**

24928 None.

24929 **SEE ALSO**

24930 *close()*, *fcntl()*, *open()*

24931 XBD <unistd.h>

24932 **CHANGE HISTORY**

24933 First released in Issue 1. Derived from Issue 1 of the SVID.

24934 **Issue 7**

24935 SD5-XSH-ERN-187 is applied.

NAME

duplocale — duplicate a locale object

SYNOPSIS

```
CX      #include <locale.h>
24940    locale_t duplocale(locale_t locobj);
```

DESCRIPTION

The *duplocale()* function shall create a duplicate copy of the locale object referenced by the *locobj* argument.

RETURN VALUE

Upon successful completion, the *duplocale()* function shall return a handle for a new locale object. Otherwise, *duplocale()* shall return **(locale_t)0** and set *errno* to indicate the error.

ERRORS

The *duplocale()* function shall fail if:

[ENOMEM] There is not enough memory available to create the locale object or load the locale data.

The *duplocale()* function may fail if:

[EINVAL] *locobj* is not a handle for a locale object.

EXAMPLES**Constructing an Altered Version of an Existing Locale Object**

The following example shows a code fragment to create a slightly altered version of an existing locale object. The function takes a locale object and a locale name and it replaces the *LC_TIME* category data in the locale object with that from the named locale.

```
24958    #include <locale.h>
24959    ...
24960    locale_t
24961    with_changed_lc_time (locale_t obj, const char *name)
24962    {
24963        locale_t retval = duplocale (obj);
24964        if (retval != (locale_t) 0)
24965        {
24966            locale_t changed = newlocale (LC_TIME_MASK, name, retval);
24967            if (changed == (locale_t) 0)
24968                /* An error occurred. Free all allocated resources. */
24969                freelocale (retval);
24970            retval = changed;
24971        }
24972        return retval; }
24973    }
```

APPLICATION USAGE

The use of the *duplocale()* function is recommended for situations where a locale object is being used in multiple places, and it is possible that the lifetime of the locale object might end before all uses are finished. Another reason to duplicate a locale object is if a slightly modified form is needed. This can be achieved by a call to *newlocale()* following the *duplocale()* call.

duplocale()*System Interfaces*

24979 As with the *newlocale()* function, handles for locale objects created by the *duplocale()* function
24980 should be released by a corresponding call to *freelocale()*.

RATIONALE

24981
24982 None.

FUTURE DIRECTIONS

24983
24984 None.

SEE ALSO

24985
24986 *freelocale()*, *newlocale()*, *uselocale()*

24987 XBD **<locale.h>**

CHANGE HISTORY

24988
24989 First released in Issue 7.

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

System Interfaces

encrypt()24990 **NAME**24991 **encrypt** — encoding function (**CRYPT**)24992 **SYNOPSIS**

```
24993 XSI      #include <unistd.h>
24994          void encrypt(char block[64], int edflag);
```

24995 **DESCRIPTION**

24996 The *encrypt()* function shall provide access to an implementation-defined encoding algorithm.
 24997 The key generated by *setkey()* is used to encrypt the string *block* with *encrypt()*.

24998 The *block* argument to *encrypt()* shall be an array of length 64 bytes containing only the bytes
 24999 with values of 0 and 1. The array is modified in place to a similar array using the key set by
 25000 *setkey()*. If *edflag* is 0, the argument is encoded. If *edflag* is 1, the argument may be decoded (see
 25001 the APPLICATION USAGE section); if the argument is not decoded, *errno* shall be set to
 25002 [ENOSYS].

25003 The *encrypt()* function shall not change the setting of *errno* if successful. An application wishing
 25004 to check for error situations should set *errno* to 0 before calling *encrypt()*. If *errno* is non-zero on
 25005 return, an error has occurred.

25006 The *encrypt()* function need not be thread-safe.

25007 **RETURN VALUE**

25008 The *encrypt()* function shall not return a value.

25009 **ERRORS**

25010 The *encrypt()* function shall fail if:

25011 [ENOSYS] The functionality is not supported on this implementation.

25012 **EXAMPLES**

25013 None.

25014 **APPLICATION USAGE**

25015 Historical implementations of the *encrypt()* function used a rather primitive encoding algorithm.

25016 In some environments, decoding might not be implemented. This is related to some Government
 25017 restrictions on encryption and decryption routines. Historical practice has been to ship a
 25018 different version of the encryption library without the decryption feature in the routines
 25019 supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

25020 **RATIONALE**

25021 None.

25022 **FUTURE DIRECTIONS**

25023 None.

25024 **SEE ALSO**

25025 *crypt()*, *setkey()*

25026 XBD <unistd.h>

25027 **CHANGE HISTORY**

25028 First released in Issue 1. Derived from Issue 1 of the SVID.

encrypt()*System Interfaces*

- 25029 **Issue 5**
25030 A note indicating that this function need not be reentrant is added to the DESCRIPTION.
- 25031 **Issue 6**
25032 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.
- 25033 **Issue 7**
25034 Austin Group Interpretation 1003.1-2001 #156 is applied.

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

NAME

endgrent, getgrent, setgrent — group database entry functions

SYNOPSIS

```

#include <grp.h>

void endgrent(void);
struct group *getgrent(void);
void setgrent(void);

```

DESCRIPTION

The *getgrent()* function shall return a pointer to a structure containing the broken-out fields of an entry in the group database. When first called, *getgrent()* shall return a pointer to a **group** structure containing the first entry in the group database. Thereafter, it shall return a pointer to a **group** structure containing the next group structure in the group database, so successive calls may be used to search the entire database.

An implementation that provides extended security controls may impose further implementation-defined restrictions on accessing the group database. In particular, the system may deny the existence of some or all of the group database entries associated with groups other than those groups associated with the caller and may omit users other than the caller from the list of members of groups in database entries that are returned.

The *setgrent()* function shall rewind the group database to allow repeated searches.

The *endgrent()* function may be called to close the group database when processing is complete.

These functions need not be thread-safe.

RETURN VALUE

When first called, *getgrent()* shall return a pointer to the first group structure in the group database. Upon subsequent calls it shall return the next group structure in the group database. The *getgrent()* function shall return a null pointer on end-of-file or an error and *errno* may be set to indicate the error.

The return value may point to a static area which is overwritten by a subsequent call to *getgrgid()*, *getgrnam()*, or *getgrent()*.

ERRORS

The *getgrent()* function may fail if:

[EINTR]	A signal was caught during the operation.
[EIO]	An I/O error has occurred.
[EMFILE]	All file descriptors available to the process are currently open.
[ENFILE]	The maximum allowable number of files is currently open in the system.

EXAMPLES

None.

APPLICATION USAGE

These functions are provided due to their historical usage. Applications should avoid dependencies on fields in the group database, whether the database is a single file, or where in the file system name space the database resides. Applications should use *getgrnam()* and *getgrgid()* whenever possible because it avoids these dependencies.

endgrent()*System Interfaces*25076 **RATIONALE**

25077 None.

25078 **FUTURE DIRECTIONS**

25079 None.

25080 **SEE ALSO**25081 *endpwent()*, *getgrgid()*, *getgrnam()*, *getlogin()*

25082 XBD <grp.h>

25083 **CHANGE HISTORY**

25084 First released in Issue 4, Version 2.

25085 **Issue 5**

25086 Moved from X/OPEN UNIX extension to BASE.

25087 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
25088 VALUE section.

25089 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

25090 **Issue 6**

25091 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

25092 **Issue 7**

25093 Austin Group Interpretation 1003.1-2001 #156 is applied.

25094 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

NAME

endhostent, gethostent, sethostent — network host database functions

SYNOPSIS

```
#include <netdb.h>

void endhostent(void);
struct hostent *gethostent(void);
void sethostent(int stayopen);
```

DESCRIPTION

These functions shall retrieve information about hosts. This information is considered to be stored in a database that can be accessed sequentially or randomly. The implementation of this database is unspecified.

Note: In many cases this database is implemented by the Domain Name System, as documented in RFC 1034, RFC 1035, and RFC 1886.

The *sethostent()* function shall open a connection to the database and set the next entry for retrieval to the first entry in the database. If the *stayopen* argument is non-zero, the connection shall not be closed by a call to *gethostent()*, and the implementation may maintain an open file descriptor.

The *gethostent()* function shall read the next entry in the database, opening and closing a connection to the database as necessary.

Entries shall be returned in **hostent** structures.

The *endhostent()* function shall close the connection to the database, releasing any open file descriptor.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, the *gethostent()* function shall return a pointer to a **hostent** structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found.

ERRORS

No errors are defined for *endhostent()*, *gethostent()*, and *sethostent()*.

EXAMPLES

None.

APPLICATION USAGE

The *gethostent()* function may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endservent()

XBD [<netdb.h>](#)

endhostent()*System Interfaces***CHANGE HISTORY**

25136 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

25138 Austin Group Interpretation 1003.1-2001 #156 is applied.

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

NAME

endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent — network database functions

SYNOPSIS

```
#include <netdb.h>

void endnetent(void);
struct netent *getnetbyaddr(uint32_t net, int type);
struct netent *getnetbyname(const char *name);
struct netent *getnetent(void);
void setnetent(int stayopen);
```

DESCRIPTION

These functions shall retrieve information about networks. This information is considered to be stored in a database that can be accessed sequentially or randomly. The implementation of this database is unspecified.

The *setnetent()* function shall open and rewind the database. If the *stayopen* argument is non-zero, the connection to the *net* database shall not be closed after each call to *getnetent()* (either directly, or indirectly through one of the other *getnet**() functions), and the implementation may maintain an open file descriptor to the database.

The *getnetent()* function shall read the next entry of the database, opening and closing a connection to the database as necessary.

The *getnetbyaddr()* function shall search the database from the beginning, and find the first entry for which the address family specified by *type* matches the *n_addrtype* member and the network number *net* matches the *n_net* member, opening and closing a connection to the database as necessary. The *net* argument shall be the network number in host byte order.

The *getnetbyname()* function shall search the database from the beginning and find the first entry for which the network name specified by *name* matches the *n_name* member, opening and closing a connection to the database as necessary.

The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* functions shall each return a pointer to a **netent** structure, the members of which shall contain the fields of an entry in the network database.

The *endnetent()* function shall close the database, releasing any open file descriptor.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* shall return a pointer to a **netent** structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer shall be returned.

ERRORS

No errors are defined.

endnetent()*System Interfaces*25178 **EXAMPLES**

25179 None.

25180 **APPLICATION USAGE**

25181 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* functions may return pointers to static data,
25182 which may be overwritten by subsequent calls to any of these functions.

25183 **RATIONALE**

25184 None.

25185 **FUTURE DIRECTIONS**

25186 None.

25187 **SEE ALSO**25188 XBD [<netdb.h>](#)25189 **CHANGE HISTORY**

25190 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

25191 **Issue 7**

25192 Austin Group Interpretation 1003.1-2001 #156 is applied.

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

System Interfaces

endprotoent()**NAME**

endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent — network protocol database functions

SYNOPSIS

```
#include <netdb.h>

void endprotoent(void);
struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);
struct protoent *getprotoent(void);
void setprotoent(int stayopen);
```

DESCRIPTION

These functions shall retrieve information about protocols. This information is considered to be stored in a database that can be accessed sequentially or randomly. The implementation of this database is unspecified.

The *setprotoent()* function shall open a connection to the database, and set the next entry to the first entry. If the *stayopen* argument is non-zero, the connection to the network protocol database shall not be closed after each call to *getprotoent()* (either directly, or indirectly through one of the other *getproto**() functions), and the implementation may maintain an open file descriptor for the database.

The *getprotobyname()* function shall search the database from the beginning and find the first entry for which the protocol name specified by *name* matches the *p_name* member, opening and closing a connection to the database as necessary.

The *getprotobynumber()* function shall search the database from the beginning and find the first entry for which the protocol number specified by *proto* matches the *p_proto* member, opening and closing a connection to the database as necessary.

The *getprotoent()* function shall read the next entry of the database, opening and closing a connection to the database as necessary.

The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions shall each return a pointer to a **protoent** structure, the members of which shall contain the fields of an entry in the network protocol database.

The *endprotoent()* function shall close the connection to the database, releasing any open file descriptor.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* return a pointer to a **protoent** structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

ERRORS

No errors are defined.

endprotoent()*System Interfaces*25233 **EXAMPLES**

25234 None.

25235 **APPLICATION USAGE**

25236 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions may return pointers to
25237 static data, which may be overwritten by subsequent calls to any of these functions.

25238 **RATIONALE**

25239 None.

25240 **FUTURE DIRECTIONS**

25241 None.

25242 **SEE ALSO**25243 XBD [*<netdb.h>*](#)25244 **CHANGE HISTORY**

25245 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

25246 **Issue 7**

25247 Austin Group Interpretation 1003.1-2001 #156 is applied.

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

25248 **NAME**25249 **endpwent, getpwent, setpwent** — user database functions25250 **SYNOPSIS**

```

25251 XSI      #include <pwd.h>
25252          void endpwent(void);
25253          struct passwd *getpwent(void);
25254          void setpwent(void);

```

25255 **DESCRIPTION**

25256 These functions shall retrieve information about users.

25257 The *getpwent()* function shall return a pointer to a structure containing the broken-out fields of
 25258 an entry in the user database. Each entry in the user database contains a **passwd** structure. When
 25259 first called, *getpwent()* shall return a pointer to a **passwd** structure containing the first entry in
 25260 the user database. Thereafter, it shall return a pointer to a **passwd** structure containing the next
 25261 entry in the user database. Successive calls can be used to search the entire user database.

25262 If an end-of-file or an error is encountered on reading, *getpwent()* shall return a null pointer.

25263 An implementation that provides extended security controls may impose further
 25264 implementation-defined restrictions on accessing the user database. In particular, the system
 25265 may deny the existence of some or all of the user database entries associated with users other
 25266 than the caller.

25267 The *setpwent()* function effectively rewinds the user database to allow repeated searches.25268 The *endpwent()* function may be called to close the user database when processing is complete.

25269 These functions need not be thread-safe.

25270 **RETURN VALUE**25271 The *getpwent()* function shall return a null pointer on end-of-file or error.25272 **ERRORS**

25273 These functions may fail if:

25274 [EIO] An I/O error has occurred.

25275 In addition, *getpwent()* and *setpwent()* may fail if:

25276 [EMFILE] All file descriptors available to the process are currently open.

25277 [ENFILE] The maximum allowable number of files is currently open in the system.

25278 The return value may point to a static area which is overwritten by a subsequent call to
 25279 *getpwuid()*, *getpwnam()*, or *getpwent()*.

endpwent()

System Interfaces

25280 **EXAMPLES**25281 **Searching the User Database**

25282 The following example uses the *getpwent()* function to get successive entries in the user
 25283 database, returning a pointer to a **passwd** structure that contains information about each user.
 25284 The call to *endpwent()* closes the user database and cleans up.

```

25285 #include <pwd.h>
25286 #include <stdio.h>

25287 void printname(uid_t uid)
25288 {
25289     struct passwd *pwd;

25290     setpwent();
25291     while((pwd = getpwent()) != NULL) {
25292         if (pwd->pw_uid == uid) {
25293             printf("name=%s\n", pwd->pw_name);
25294             break;
25295         }
25296     }
25297     endpwent();
25298 }
```

25299 **APPLICATION USAGE**

25300 These functions are provided due to their historical usage. Applications should avoid
 25301 dependencies on fields in the password database, whether the database is a single file, or where
 25302 in the file system name space the database resides. Applications should use *getpwuid()*
 25303 whenever possible because it avoids these dependencies.

25304 **RATIONALE**

25305 None.

25306 **FUTURE DIRECTIONS**

25307 None.

25308 **SEE ALSO**

25309 *endgrent()*, *getlogin()*, *getpwnam()*, *getpwuid()*

25310 XBD **<pwd.h>**

25311 **CHANGE HISTORY**

25312 First released in Issue 4, Version 2.

25313 **Issue 5**

25314 Moved from X/OPEN UNIX extension to BASE.

25315 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
 25316 VALUE section.

25317 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

25318 **Issue 6**

25319 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

Issue 7

25320 Austin Group Interpretation 1003.1-2001 #156 is applied.

25321 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

25322 The EXAMPLES section is revised.

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

endservent()*System Interfaces*25324 **NAME**

25325 endservent, getservbyname, getservbyport, getservent, setservent — network services database
 25326 functions

25327 **SYNOPSIS**

```
25328 #include <netdb.h>
25329
25329 void endservent(void);
25330 struct servent *getservbyname(const char *name, const char *proto);
25331 struct servent *getservbyport(int port, const char *proto);
25332 struct servent *getservent(void);
25333 void setservent(int stayopen);
```

25334 **DESCRIPTION**

25335 These functions shall retrieve information about network services. This information is
 25336 considered to be stored in a database that can be accessed sequentially or randomly. The
 25337 implementation of this database is unspecified.

25338 The *setservent()* function shall open a connection to the database, and set the next entry to the
 25339 first entry. If the *stayopen* argument is non-zero, the *net* database shall not be closed after each
 25340 call to the *getservent()* function (either directly, or indirectly through one of the other *getserv**(
 25341 functions), and the implementation may maintain an open file descriptor for the database.

25342 The *getservent()* function shall read the next entry of the database, opening and closing a
 25343 connection to the database as necessary.

25344 The *getservbyname()* function shall search the database from the beginning and find the first
 25345 entry for which the service name specified by *name* matches the *s_name* member and the protocol
 25346 name specified by *proto* matches the *s_proto* member, opening and closing a connection to the
 25347 database as necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be
 25348 matched.

25349 The *getservbyport()* function shall search the database from the beginning and find the first entry
 25350 for which the port specified by *port* matches the *s_port* member and the protocol name specified
 25351 by *proto* matches the *s_proto* member, opening and closing a connection to the database as
 25352 necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be matched. The *port*
 25353 argument shall be a value obtained by converting a **uint16_t** in network byte order to **int**.

25354 The *getservbyname()*, *getservbyport()*, and *getservent()* functions shall each return a pointer to a
 25355 **servent** structure, the members of which shall contain the fields of an entry in the network
 25356 services database.

25357 The *endservent()* function shall close the database, releasing any open file descriptor.

25358 These functions need not be thread-safe.

25359 **RETURN VALUE**

25360 Upon successful completion, *getservbyname()*, *getservbyport()*, and *getservent()* return a pointer to
 25361 a **servent** structure if the requested entry was found, and a null pointer if the end of the database
 25362 was reached or the requested entry was not found. Otherwise, a null pointer is returned.

25363 **ERRORS**

25364 No errors are defined.

25365 **EXAMPLES**

25366 None.

25367 **APPLICATION USAGE**25368 The *port* argument of *getservbyport()* need not be compatible with the port values of all address
25369 families.25370 The *getservbyname()*, *getservbyport()*, and *getservent()* functions may return pointers to static
25371 data, which may be overwritten by subsequent calls to any of these functions.25372 **RATIONALE**

25373 None.

25374 **FUTURE DIRECTIONS**

25375 None.

25376 **SEE ALSO**25377 *endhostent()*, *endprotoent()*, *htonl()*, *inet_addr()*

25378 XBD <netdb.h>

25379 **CHANGE HISTORY**

25380 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

25381 **Issue 7**

25382 Austin Group Interpretation 1003.1-2001 #156 is applied.

25383 SD5-XBD-ERN-14 is applied.

endutxent()*System Interfaces*25384 **NAME**

25385 **endutxent**, **getutxent**, **getutxid**, **getutxline**, **pututxline**, **setutxent** — user accounting database
 25386 functions

25387 **SYNOPSIS**

```
25388 XSI      #include <utmpx.h>
25389
25389      void endutxent(void);
25390      struct utmpx *getutxent(void);
25391      struct utmpx *getutxid(const struct utmpx *id);
25392      struct utmpx *getutxline(const struct utmpx *line);
25393      struct utmpx *pututxline(const struct utmpx *utmpx);
25394      void setutxent(void);
```

25395 **DESCRIPTION**

25396 These functions shall provide access to the user accounting database.

25397 The *getutxent()* function shall read the next entry from the user accounting database. If the
 25398 database is not already open, it shall open it. If it reaches the end of the database, it shall fail.

25399 The *getutxid()* function shall search forward from the current point in the database. If the
 25400 *ut_type* value of the **utmpx** structure pointed to by *id* is **BOOT_TIME**, **OLD_TIME**, or
 25401 **NEW_TIME**, then it shall stop when it finds an entry with a matching *ut_type* value. If the
 25402 *ut_type* value is **INIT_PROCESS**, **LOGIN_PROCESS**, **USER_PROCESS**, or **DEAD_PROCESS**,
 25403 then it shall stop when it finds an entry whose type is one of these four and whose *ut_id* member
 25404 matches the *ut_id* member of the **utmpx** structure pointed to by *id*. If the end of the database is
 25405 reached without a match, *getutxid()* shall fail.

25406 The *getutxline()* function shall search forward from the current point in the database until it
 25407 finds an entry of the type **LOGIN_PROCESS** or **USER_PROCESS** which also has a *ut_line* value
 25408 matching that in the **utmpx** structure pointed to by *line*. If the end of the database is reached
 25409 without a match, *getutxline()* shall fail.

25410 The *getutxid()* or *getutxline()* function may cache data. For this reason, to use *getutxline()* to
 25411 search for multiple occurrences, the application shall zero out the static data after each success,
 25412 or *getutxline()* may return a pointer to the same **utmpx** structure.

25413 There is one exception to the rule about clearing the structure before further reads are done. The
 25414 implicit read done by *pututxline()* (if it finds that it is not already at the correct place in the user
 25415 accounting database) shall not modify the static structure returned by *getutxent()*, *getutxid()*, or
 25416 *getutxline()*, if the application has modified this structure and passed the pointer back to
 25417 *pututxline()*.

25418 For all entries that match a request, the *ut_type* member indicates the type of the entry. Other
 25419 members of the entry shall contain meaningful data based on the value of the *ut_type* member as
 25420 follows:

ut_type Member	Other Members with Meaningful Data
EMPTY	No others
BOOT_TIME	<i>ut_tv</i>
OLD_TIME	<i>ut_tv</i>
NEW_TIME	<i>ut_tv</i>
USER_PROCESS	<i>ut_id</i> , <i>ut_user</i> (login name of the user), <i>ut_line</i> , <i>ut_pid</i> , <i>ut_tv</i>
INIT_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>
LOGIN_PROCESS	<i>ut_id</i> , <i>ut_user</i> (implementation-defined name of the login process), <i>ut_pid</i> , <i>ut_tv</i>
DEAD_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>

An implementation that provides extended security controls may impose implementation-defined restrictions on accessing the user accounting database. In particular, the system may deny the existence of some or all of the user accounting database entries associated with users other than the caller.

If the process has appropriate privileges, the *pututxline()* function shall write out the structure into the user accounting database. It shall use *getutxid()* to search for a record that satisfies the request. If this search succeeds, then the entry shall be replaced. Otherwise, a new entry shall be made at the end of the user accounting database.

The *endutxent()* function shall close the user accounting database.

The *setutxent()* function shall reset the input to the beginning of the database. This should be done before each search for a new entry if it is desired that the entire database be examined.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, *getutxent()*, *getutxid()*, and *getutxline()* shall return a pointer to a **utmpx** structure containing a copy of the requested entry in the user accounting database. Otherwise, a null pointer shall be returned.

The return value may point to a static area which is overwritten by a subsequent call to *getutxid()* or *getutxline()*.

Upon successful completion, *pututxline()* shall return a pointer to a **utmpx** structure containing a copy of the entry added to the user accounting database. Otherwise, a null pointer shall be returned.

The *endutxent()* and *setutxent()* functions shall not return a value.

ERRORS

No errors are defined for the *endutxent()*, *getutxent()*, *getutxid()*, *getutxline()*, and *setutxent()* functions.

The *pututxline()* function may fail if:

[EPERM] The process does not have appropriate privileges.

endutxent()*System Interfaces*25458 **EXAMPLES**

25459 None.

25460 **APPLICATION USAGE**25461 The sizes of the arrays in the structure can be found using the *sizeof* operator.25462 **RATIONALE**

25463 None.

25464 **FUTURE DIRECTIONS**

25465 None.

25466 **SEE ALSO**25467 XBD [<utmpx.h>](#)25468 **CHANGE HISTORY**

25469 First released in Issue 4, Version 2.

25470 **Issue 5**

25471 Moved from X/OPEN UNIX extension to BASE.

25472 Normative text previously in the APPLICATION USAGE section is moved to the
25473 DESCRIPTION.

25474 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

25475 **Issue 6**

25476 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

25477 **Issue 7**

25478 Austin Group Interpretation 1003.1-2001 #156 is applied.

*System Interfaces***environ**25479 **NAME**

25480 `environ` — array of character pointers to the environment strings

25481 **SYNOPSIS**

25482 `extern char **environ;`

25483 **DESCRIPTION**

25484 Refer to *exec* and XBD Chapter 8 (on page 173).

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

erand48()*System Interfaces*25485 **NAME**25486 **erand48** — generate uniformly distributed pseudo-random numbers25487 **SYNOPSIS**25488 XSI `#include <stdlib.h>`25489 `double erand48(unsigned short xsubi[3]);`25490 **DESCRIPTION**25491 Refer to *drand48()*.

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

NAME

erf, erff, erfl — error functions

SYNOPSIS

```
#include <math.h>

double erf(double x);
float erff(float x);
long double erfl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall compute the error function of their argument x , defined as:

$$\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the value of the error function.

MX

If x is NaN, a NaN shall be returned.

If x is ± 0 , ± 0 shall be returned.

If x is $\pm \text{Inf}$, ± 1 shall be returned.

If x is subnormal, a range error may occur, and $2 * x / \text{sqrt}(\pi)$ should be returned.

ERRORS

These functions may fail if:

MX

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES**Computing the Probability for a Normal Variate**

This example shows how to use *erf()* to compute the probability that a normal variate assumes a value in the range $[x1, x2]$ with $x1 \leq x2$.

This example uses the constant M_SQRT1_2 which is part of the XSI option.

```
#include <math.h>

double
Phi(const double x1, const double x2)
{
    return ( erf(x2*M_SQRT1_2) - erf(x1*M_SQRT1_2) ) / 2;
```

erf()*System Interfaces*

25532 }

25533 **APPLICATION USAGE**

25534 Underflow occurs when $|x| < \text{DBL\_MIN} * (\text{sqrt}(\pi)/2)$.

25535 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
25536 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

25537 **RATIONALE**

25538 None.

25539 **FUTURE DIRECTIONS**

25540 None.

25541 **SEE ALSO**

25542 *erfc()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

25543 XBD Section 4.19 (on page 116), **<math.h>**

25544 **CHANGE HISTORY**

25545 First released in Issue 1. Derived from Issue 1 of the SVID.

25546 **Issue 5**

25547 The DESCRIPTION is updated to indicate how an application should check for an error. This
25548 text was previously published in the APPLICATION USAGE section.

25549 **Issue 6**

25550 The *erf()* function is no longer marked as an extension.

25551 The *erfc()* function is split out onto its own reference page.

25552 The *erff()* and *erfl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

25553 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
25554 revised to align with the ISO/IEC 9899:1999 standard.

25555 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
25556 marked.

25557 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/22 is applied, adding the example to the
25558 EXAMPLES section.

25559 **NAME**25560 `erfc, erfcf, erfcl` — complementary error functions25561 **SYNOPSIS**

```
25562     #include <math.h>

25563     double erfc(double x);
25564     float erfcf(float x);
25565     long double erfcl(long double x);
```

25566 **DESCRIPTION**

25567 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 25568 conflict between the requirements described here and the ISO C standard is unintentional. This
 25569 volume of POSIX.1-2008 defers to the ISO C standard.

25570 These functions shall compute the complementary error function $1.0 - \text{erf}(x)$.

25571 An application wishing to check for error situations should set `errno` to zero and call
 25572 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or
 25573 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-
 25574 zero, an error has occurred.

25575 **RETURN VALUE**

25576 Upon successful completion, these functions shall return the value of the complementary error
 25577 function.

25578 If the correct value would cause underflow and is not representable, a range error may occur
 25579 **MX** and either 0.0 (if representable), or an implementation-defined value shall be returned.

25580 **MX** If x is NaN, a NaN shall be returned.

25581 If x is ± 0 , +1 shall be returned.

25582 If x is $-\text{Inf}$, +2 shall be returned.

25583 If x is $+\text{Inf}$, +0 shall be returned.

25584 If the correct value would cause underflow and is representable, a range error may occur and
 25585 the correct value shall be returned.

25586 **ERRORS**

25587 These functions may fail if:

25588 **Range Error** The result underflows.

25589 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 25590 then `errno` shall be set to [ERANGE]. If the integer expression
 25591 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the underflow
 25592 floating-point exception shall be raised.

25593 **EXAMPLES**

25594 None.

25595 **APPLICATION USAGE**

25596 The `erfc()` function is provided because of the extreme loss of relative accuracy if `erf(x)` is called
 25597 for large x and the result subtracted from 1.0.

25598 Note for IEEE Std 754-1985 **double**, $26.55 < x$ implies `erfc(x)` has underflowed.

25599 On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling & MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.
 25600

erfc()*System Interfaces*25601 **RATIONALE**

25602 None.

25603 **FUTURE DIRECTIONS**

25604 None.

25605 **SEE ALSO**25606 *erf()*, *feclearexcept()*, *fetestexcept()*, *isnan()*25607 XBD Section 4.19 (on page 116), **<math.h>**25608 **CHANGE HISTORY**

25609 First released in Issue 1. Derived from Issue 1 of the SVID.

25610 **Issue 5**25611 The DESCRIPTION is updated to indicate how an application should check for an error. This
25612 text was previously published in the APPLICATION USAGE section.25613 **Issue 6**25614 The *erfc()* function is no longer marked as an extension.25615 These functions are split out from the *erf()* reference page.25616 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
25617 revised to align with the ISO/IEC 9899:1999 standard.25618 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
25619 marked.

*System Interfaces***erff()**25620 **NAME**

25621 erff, erfl — error functions

25622 **SYNOPSIS**

25623 #include <math.h>

25624 float erff(float x);

25625 long double erfl(long double x);

25626 **DESCRIPTION**25627 Refer to *erf()*.

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

errno*System Interfaces*25628 **NAME**25629 **errno** — error return value25630 **SYNOPSIS**25631 `#include <errno.h>`25632 **DESCRIPTION**25633 The lvalue *errno* is used by many functions to return error values.

25634 Many functions provide an error number in *errno*, which has type **int** and is defined in
 25635 **<errno.h>**. The value of *errno* shall be defined only after a call to a function for which it is
 25636 explicitly stated to be set and until it is changed by the next function call or if the application
 25637 assigns it a value. The value of *errno* should only be examined when it is indicated to be valid by
 25638 a function's return value. Applications shall obtain the definition of *errno* by the inclusion of
 25639 **<errno.h>**. No function in this volume of POSIX.1-2008 shall set *errno* to 0. The setting of *errno*
 25640 after a successful call to a function is unspecified unless the description of that function specifies
 25641 that *errno* shall not be modified.

25642 It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a
 25643 macro definition is suppressed in order to access an actual object, or a program defines an
 25644 identifier with the name *errno*, the behavior is undefined.

25645 The symbolic values stored in *errno* are documented in the ERRORS sections on all relevant
 25646 pages.

25647 **RETURN VALUE**

25648 None.

25649 **ERRORS**

25650 None.

25651 **EXAMPLES**

25652 None.

25653 **APPLICATION USAGE**

25654 Previously both POSIX and X/Open documents were more restrictive than the ISO C standard
 25655 in that they required *errno* to be defined as an external variable, whereas the ISO C standard
 25656 required only that *errno* be defined as a modifiable lvalue with type **int**.

25657 An application that needs to examine the value of *errno* to determine the error should set it to 0
 25658 before a function call, then inspect it before a subsequent function call.

25659 **RATIONALE**

25660 None.

25661 **FUTURE DIRECTIONS**

25662 None.

25663 **SEE ALSO**25664 [Section 2.3](#)25665 XBD **<errno.h>**25666 **CHANGE HISTORY**

25667 First released in Issue 1. Derived from Issue 1 of the SVID.

25668 **Issue 5**

25669 The following sentence is deleted from the DESCRIPTION: "The value of *errno* is 0 at program
 25670 start-up, but is never set to 0 by any XSI function".

25671 The DESCRIPTION also no longer states that conforming implementations may support the
25672 declaration:

25673 `extern int errno;`

25674 **Issue 6**

25675 Obsolescent text regarding defining *errno* as:

25676 `extern int errno`

25677 is removed.

25678 Text regarding no function setting *errno* to zero to indicate an error is changed to no function
25679 shall set *errno* to zero. This is for alignment with the ISO/IEC 9899:1999 standard.

25680 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/23 is applied, adding text to the
25681 DESCRIPTION stating that the setting of *errno* after a successful call to a function is unspecified
25682 unless the description of the function requires that it will not be modified.

exec*System Interfaces***NAME**

`environ`, `execl`, `execle`, `execlp`, `execv`, `execve`, `execvp`, `fexecve` — execute a file

SYNOPSIS

```
#include <unistd.h>

extern char **environ;

int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
int execle(const char *path, const char *arg0, ... /*,
           (char *)0, char *const envp[] */);
int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
int execv(const char *path, char *const argv[]);
int execve(const char *path, char *const argv[], char *const envp[]);
int execvp(const char *file, char *const argv[]);
int fexecve(int fd, char *const argv[], char *const envp[]);
```

DESCRIPTION

The *exec* family of functions shall replace the current process image with a new process image. The new image shall be constructed from a regular, executable file called the *new process image file*. There shall be no return from a successful *exec*, because the calling process image is overlaid by the new process image.

The *fexecve()* function shall be equivalent to the *execve()* function except that the file to be executed is determined by the file descriptor *fd* instead of a pathname. The file offset of *fd* is ignored.

When a C-language program is executed as a result of a call to one of the *exec* family of functions, it shall be entered as a C-language function call as follows:

```
int main (int argc, char *argv[]);
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. In addition, the following variable:

```
extern char **environ;
```

is initialized as a pointer to an array of character pointers to the environment strings. The *argv* and *environ* arrays are each terminated by a null pointer. The null pointer terminating the *argv* array is not counted in *argc*.

Conforming multi-threaded applications shall not use the *environ* variable to access or modify any environment variable while any other thread is concurrently modifying any environment variable. A call to any function dependent on any environment variable shall be considered a use of the *environ* variable to access that environment variable.

The arguments specified by a program with one of the *exec* functions shall be passed on to the new process image in the corresponding *main()* arguments.

The argument *path* points to a pathname that identifies the new process image file.

The argument *file* is used to construct a pathname that identifies the new process image file. If the *file* argument contains a <slash> character, the *file* argument shall be used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed as the environment variable *PATH* (see XBD [Chapter 8](#), on page 173). If this environment variable is not present, the results of the search are implementation-defined.

There are two distinct ways in which the contents of the process image file may cause the execution to fail, distinguished by the setting of *errno* to either [ENOEXEC] or [EINVAL] (see the **ERRORS** section). In the cases where the other members of the *exec* family of functions would

System Interfaces

exec

fail and set *errno* to [ENOEXEC], the *execlp()* and *execvp()* functions shall execute a command interpreter and the environment of the executed command shall be as if the process invoked the *sh* utility using *execl()* as follows:

```
execl(<shell path>, arg0, file, arg1, ..., (char *)0);
```

where *<shell path>* is an unspecified pathname for the *sh* utility, *file* is the process image file, and for *execvp()*, where *arg0*, *arg1*, and so on correspond to the values passed to *execvp()* in *argv[0]*, *argv[1]*, and so on.

The arguments represented by *arg0*, ... are pointers to null-terminated character strings. These strings shall constitute the argument list available to the new process image. The list is terminated by a null pointer. The argument *arg0* should point to a filename that is associated with the process being started by one of the *exec* functions.

The argument *argv* is an array of character pointers to null-terminated strings. The application shall ensure that the last member of this array is a null pointer. These strings shall constitute the argument list available to the new process image. The value in *argv[0]* should point to a filename that is associated with the process being started by one of the *exec* functions.

The argument *envp* is an array of character pointers to null-terminated strings. These strings shall constitute the environment for the new process image. The *envp* array is terminated by a null pointer.

For those forms not containing an *envp* pointer (*execl()*, *execv()*, *execlp()*, and *execvp()*), the environment for the new process image shall be taken from the external variable *environ* in the calling process.

The number of bytes available for the new process' combined argument and environment lists is {ARG_MAX}. It is implementation-defined whether null terminators, pointers, and/or any alignment bytes are included in this total.

File descriptors open in the calling process image shall remain open in the new process image, except for those whose close-on-exec flag FD_CLOEXEC is set. For those file descriptors that remain open, all attributes of the open file description remain unchanged. For any file descriptor that is closed for this reason, file locks are removed as a result of the close as described in *close()*. Locks that are not removed by closing of file descriptors remain unchanged.

If file descriptors 0, 1, and 2 would otherwise be closed after a successful call to one of the *exec* family of functions, and the new process image file has the set-user-ID or set-group-ID file mode bits set, and the ST_NOSUID bit is not set for the file system containing the new process image file, implementations may open an unspecified file for each of these file descriptors in the new process image.

Directory streams open in the calling process image shall be closed in the new process image.

The state of the floating-point environment in the initial thread of the new process image shall be set to the default.

The state of conversion descriptors and message catalog descriptors in the new process image is undefined.

For the new process image, the equivalent of:

```
setlocale(LC_ALL, "C")
```

shall be executed at start-up.

Signals set to the default action (SIG_DFL) in the calling process image shall be set to the default action in the new process image. Except for SIGCHLD, signals set to be ignored (SIG_IGN) by

exec*System Interfaces*

25772		the calling process image shall be set to be ignored by the new process image. Signals set to be
25773		caught by the calling process image shall be set to the default action in the new process image
25774		(see <signal.h>).
25775		If the SIGCHLD signal is set to be ignored by the calling process image, it is unspecified whether
25776		the SIGCHLD signal is set to be ignored or to the default action in the new process image.
25777	XSI	After a successful call to any of the <i>exec</i> functions, alternate signal stacks are not preserved and
25778		the SA_ONSTACK flag shall be cleared for all signals.
25779		After a successful call to any of the <i>exec</i> functions, any functions previously registered by the
25780		<i>atexit()</i> or <i>pthread_atfork()</i> functions are no longer registered.
25781	XSI	If the ST_NOSUID bit is set for the file system containing the new process image file, then the
25782		effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged
25783		in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is
25784		set, the effective user ID of the new process image shall be set to the user ID of the new process
25785		image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the
25786		effective group ID of the new process image shall be set to the group ID of the new process
25787		image file. The real user ID, real group ID, and supplementary group IDs of the new process
25788		image shall remain the same as those of the calling process image. The effective user ID and
25789		effective group ID of the new process image shall be saved (as the saved set-user-ID and the
25790		saved set-group-ID) for use by <i>setuid()</i> .
25791	XSI	Any shared memory segments attached to the calling process image shall not be attached to the
25792		new process image.
25793		Any named semaphores open in the calling process shall be closed as if by appropriate calls to
25794		<i>sem_close()</i> .
25795	TYM	Any blocks of typed memory that were mapped in the calling process are unmapped, as if
25796		<i>munmap()</i> was implicitly called to unmap them.
25797	ML	Memory locks established by the calling process via calls to <i>mlockall()</i> or <i>mlock()</i> shall be
25798		removed. If locked pages in the address space of the calling process are also mapped into the
25799		address spaces of other processes and are locked by those processes, the locks established by the
25800		other processes shall be unaffected by the call by this process to the <i>exec</i> function. If the <i>exec</i>
25801		function fails, the effect on memory locks is unspecified.
25802		Memory mappings created in the process are unmapped before the address space is rebuilt for
25803		the new process image.
25804	SS	When the calling process image does not use the SCHED_FIFO, SCHED_RR, or
25805		SCHED_SPORADIC scheduling policies, the scheduling policy and parameters of the new
25806		process image and the initial thread in that new process image are implementation-defined.
25807	PS	When the calling process image uses the SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC
25808		scheduling policies, the process policy and scheduling parameter settings shall not be changed
25809	TPS	by a call to an <i>exec</i> function. The initial thread in the new process image shall inherit the process
25810		scheduling policy and parameters. It shall have the default system contention scope, but shall
25811		inherit its allocation domain from the calling process image.
25812		Per-process timers created by the calling process shall be deleted before replacing the current
25813		process image with the new process image.
25814	MSG	All open message queue descriptors in the calling process shall be closed, as described in
25815		<i>mq_close()</i> .
25816		Any outstanding asynchronous I/O operations may be canceled. Those asynchronous I/O

25817		operations that are not canceled shall complete as if the <i>exec</i> function had not yet occurred, but
25818		any associated signal notifications shall be suppressed. It is unspecified whether the <i>exec</i>
25819		function itself blocks awaiting such I/O completion. In no event, however, shall the new process
25820		image created by the <i>exec</i> function be affected by the presence of outstanding asynchronous I/O
25821		operations at the time the <i>exec</i> function is called. Whether any I/O is canceled, and which I/O
25822		may be canceled upon <i>exec</i> , is implementation-defined.
25823	CPT	The new process image shall inherit the CPU-time clock of the calling process image. This
25824		inheritance means that the process CPU-time clock of the process being <i>exec</i> -ed shall not be
25825		reinitialized or altered as a result of the <i>exec</i> function other than to reflect the time spent by the
25826		process executing the <i>exec</i> function itself.
25827	TCT	The initial value of the CPU-time clock of the initial thread of the new process image shall be set
25828		to zero.
25829	OB TRC	If the calling process is being traced, the new process image shall continue to be traced into the
25830		same trace stream as the original process image, but the new process image shall not inherit the
25831		mapping of trace event names to trace event type identifiers that was defined by calls to the
25832		<i>posix_trace_eventid_open()</i> or the <i>posix_trace_trid_eventid_open()</i> functions in the calling process
25833		image.
25834		If the calling process is a trace controller process, any trace streams that were created by the
25835		calling process shall be shut down as described in the <i>posix_trace_shutdown()</i> function.
25836		The thread ID of the initial thread in the new process image is unspecified.
25837		The size and location of the stack on which the initial thread in the new process image runs is
25838		unspecified.
25839		The initial thread in the new process image shall have its cancellation type set to
25840		PTHREAD_CANCEL_DEFERRED and its cancellation state set to
25841		PTHREAD_CANCEL_ENABLED.
25842		The initial thread in the new process image shall have all thread-specific data values set to
25843		NULL and all thread-specific data keys shall be removed by the call to <i>exec</i> without running
25844		destructors.
25845		The initial thread in the new process image shall be joinable, as if created with the <i>detachstate</i>
25846		attribute set to PTHREAD_CREATE_JOINABLE.
25847		The new process shall inherit at least the following attributes from the calling process image:
25848	XSI	• Nice value (see <i>nice()</i>)
25849	XSI	• <i>semadj</i> values (see <i>semop()</i>)
25850		• Process ID
25851		• Parent process ID
25852		• Process group ID
25853		• Session membership
25854		• Real user ID
25855		• Real group ID
25856		• Supplementary group IDs

25857		• Time left until an alarm clock signal (see <i>alarm()</i>)
25858		• Current working directory
25859		• Root directory
25860		• File mode creation mask (see <i>umask()</i>)
25861	XSI	• File size limit (see <i>getrlimit()</i> and <i>setrlimit()</i>)
25862		• Process signal mask (see <i>pthread_sigmask()</i>)
25863		• Pending signal (see <i>sigpending()</i>)
25864		• <i>tms_utime</i> , <i>tms_stime</i> , <i>tms_cutime</i> , and <i>tms_cstime</i> (see <i>times()</i>)
25865	XSI	• Resource limits
25866		• Controlling terminal
25867	XSI	• Interval timers
25868		The initial thread of the new process shall inherit at least the following attributes from the
25869		calling thread:
25870		• Signal mask (see <i>sigprocmask()</i> and <i>pthread_sigmask()</i>)
25871		• Pending signals (see <i>sigpending()</i>)
25872		All other process attributes defined in this volume of POSIX.1-2008 shall be inherited in the new
25873		process image from the old process image. All other thread attributes defined in this volume of
25874		POSIX.1-2008 shall be inherited in the initial thread in the new process image from the calling
25875		thread in the old process image. The inheritance of process or thread attributes not defined by
25876		this volume of POSIX.1-2008 is implementation-defined.
25877		A call to any <i>exec</i> function from a process with more than one thread shall result in all threads
25878		being terminated and the new executable image being loaded and executed. No destructor
25879		functions or cleanup handlers shall be called.
25880		Upon successful completion, the <i>exec</i> functions shall mark for update the last data access
25881		timestamp of the file. If an <i>exec</i> function failed but was able to locate the process image file,
25882		whether the last data access timestamp is marked for update is unspecified. Should the <i>exec</i>
25883		function succeed, the process image file shall be considered to have been opened with <i>open()</i> .
25884		The corresponding <i>close()</i> shall be considered to occur at a time after this open, but before
25885		process termination or successful completion of a subsequent call to one of the <i>exec</i> functions,
25886		<i>posix_spawn()</i> , or <i>posix_spawnnp()</i> . The <i>argv[]</i> and <i>envp[]</i> arrays of pointers and the strings to
25887		which those arrays point shall not be modified by a call to one of the <i>exec</i> functions, except as a
25888		consequence of replacing the process image.
25889	XSI	The saved resource limits in the new process image are set to be a copy of the process'
25890		corresponding hard and soft limits.
25891		RETURN VALUE
25892		If one of the <i>exec</i> functions returns to the calling process image, an error has occurred; the return
25893		value shall be <i>-1</i> , and <i>errno</i> shall be set to indicate the error.
25894		ERRORS
25895		The <i>exec</i> functions shall fail if:
25896	[E2BIG]	The number of bytes used by the new process image's argument list and
25897		environment list is greater than the system-imposed limit of {ARG_MAX}
25898		bytes.

25899	[EACCES]	Search permission is denied for a directory listed in the new process image file's path prefix, or the new process image file denies execution permission, or the new process image file is not a regular file and the implementation does not support execution of files of its type.
25900		
25901		
25902		
25903	[EINVAL]	The new process image file has appropriate privileges and has a recognized executable binary format, but the system does not support execution of a file with this format.
25904		
25905		
25906	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> or <i>file</i> argument.
25907		
25908	[ENAMETOOLONG]	
25909		The length of a component of a pathname is longer than {NAME_MAX}.
25910	[ENOENT]	A component of <i>path</i> or <i>file</i> does not name an existing file or <i>path</i> or <i>file</i> is an empty string.
25911		
25912	[ENOTDIR]	A component of the new process image file's path prefix is not a directory, or the new process image file's pathname contains at least one non- <code><slash></code> character and ends with one or more trailing <code><slash></code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
25913		
25914		
25915		
25916		
25917		The <i>exec</i> functions, except for <i>execlp()</i> and <i>execvp()</i> , shall fail if:
25918	[ENOEXEC]	The new process image file has the appropriate access permission but has an unrecognized format.
25919		
25920		The <i>fexecve()</i> function shall fail if:
25921	[EBADF]	The <i>fd</i> argument is not a valid file descriptor open for executing.
25922		The <i>exec</i> functions may fail if:
25923	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> or <i>file</i> argument.
25924		
25925	[ENAMETOOLONG]	
25926		The length of the <i>path</i> argument or the length of the pathname constructed from the <i>file</i> argument exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
25927		
25928		
25929		
25930	[ENOMEM]	The new process image requires more memory than is allowed by the hardware or system-imposed memory management constraints.
25931		
25932	[ETXTBSY]	The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.
25933		

25934 EXAMPLES

25935 Using execl()

25936 The following example executes the *ls* command, specifying the pathname of the executable
25937 (*/bin/ls*) and using arguments supplied directly to the command to produce single-column
25938 output.

```
25939 #include <unistd.h>
25940 int ret;
25941 ...
25942 ret = execl ("/bin/ls", "ls", "-l", (char *)0);
```

25943 Using execl_e()

25944 The following example is similar to [Using execl\(\)](#). In addition, it specifies the environment for
25945 the new process image using the *env* argument.

```
25946 #include <unistd.h>
25947 int ret;
25948 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
25949 ...
25950 ret = execl_e ("/bin/ls", "ls", "-l", (char *)0, env);
```

25951 Using execl_p()

25952 The following example searches for the location of the *ls* command among the directories
25953 specified by the *PATH* environment variable.

```
25954 #include <unistd.h>
25955 int ret;
25956 ...
25957 ret = execl_p ("ls", "ls", "-l", (char *)0);
```

25958 Using execv()

25959 The following example passes arguments to the *ls* command in the *cmd* array.

```
25960 #include <unistd.h>
25961 int ret;
25962 char *cmd[] = { "ls", "-l", (char *)0 };
25963 ...
25964 ret = execv ("/bin/ls", cmd);
```


Using `execve()`

The following example passes arguments to the `ls` command in the `cmd` array, and specifies the environment for the new process image using the `env` argument.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
...
ret = execve ("/bin/ls", cmd, env);
```

Using `execvp()`

The following example searches for the location of the `ls` command among the directories specified by the `PATH` environment variable, and passes arguments to the `ls` command in the `cmd` array.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
...
ret = execvp ("ls", cmd);
```

APPLICATION USAGE

As the state of conversion descriptors and message catalog descriptors in the new process image is undefined, conforming applications should not rely on their use and should close them prior to calling one of the `exec` functions.

Applications that require other than the default POSIX locale should call `setlocale()` with the appropriate parameters to establish the locale of the new process.

The `environ` array should not be accessed directly by the application.

The new process might be invoked in a non-conforming environment if the `envp` array does not contain implementation-defined variables required by the implementation to provide a conforming environment. See the `_CS_V7_ENV` entry in `<unistd.h>` and `confstr()` for details.

Applications should not depend on file descriptors 0, 1, and 2 being closed after an `exec`. A future version may allow these file descriptors to be automatically opened for any process.

If an application wants to perform a checksum test of the file being executed before executing it, the file will need to be opened with read permission to perform the checksum test.

Since execute permission is checked by `fexecve()`, the file description `fd` need not have been opened with the `O_EXEC` flag. However, if the file to be executed denies read and write permission for the process preparing to do the `exec`, the only way to provide the `fd` to `fexecve()` will be to use the `O_EXEC` flag when opening `fd`. In this case, the application will not be able to perform a checksum test since it will not be able to read the contents of the file.

Note that when a file descriptor is opened with `O_RDONLY`, `O_RDWR`, or `O_WRONLY` mode, the file descriptor can be used to read, read and write, or write the file, respectively, even if the mode of the file changes after the file was opened. Using the `O_EXEC` open mode is different; `fexecve()` will ignore the mode that was used when the file descriptor was opened and the `exec` will fail if the mode of the file associated with `fd` does not grant execute permission to the calling process at the time `fexecve()` is called.

RATIONALE

Early proposals required that the value of *argc* passed to *main()* be “one or greater”. This was driven by the same requirement in drafts of the ISO C standard. In fact, historical implementations have passed a value of zero when no arguments are supplied to the caller of the *exec* functions. This requirement was removed from the ISO C standard and subsequently removed from this volume of POSIX.1-2008 as well. The wording, in particular the use of the word *should*, requires a Strictly Conforming POSIX Application to pass at least one argument to the *exec* function, thus guaranteeing that *argc* be one or greater when invoked by such an application. In fact, this is good practice, since many existing applications reference *argv*[0] without first checking the value of *argc*.

The requirement on a Strictly Conforming POSIX Application also states that the value passed as the first argument be a filename associated with the process being started. Although some existing applications pass a pathname rather than a filename in some circumstances, a filename is more generally useful, since the common usage of *argv*[0] is in printing diagnostics. In some cases the filename passed is not the actual filename of the file; for example, many implementations of the *login* utility use a convention of prefixing a hyphen (<code>' - '</code>) to the actual filename, which indicates to the command interpreter being invoked that it is a “login shell”.

Historically, there have been two ways that implementations can *exec* shell scripts.

One common historical implementation is that the *execl()*, *execv()*, *execle()*, and *execve()* functions return an [ENOEXEC] error for any file not recognizable as executable, including a shell script. When the *execlp()* and *execvp()* functions encounter such a file, they assume the file to be a shell script and invoke a known command interpreter to interpret such files. This is now required by POSIX.1-2008. These implementations of *execvp()* and *execlp()* only give the [ENOEXEC] error in the rare case of a problem with the command interpreter’s executable file. Because of these implementations, the [ENOEXEC] error is not mentioned for *execlp()* or *execvp()*, although implementations can still give it.

Another way that some historical implementations handle shell scripts is by recognizing the first two bytes of the file as the character string “#!” and using the remainder of the first line of the file as the name of the command interpreter to execute.

One potential source of confusion noted by the standard developers is over how the contents of a process image file affect the behavior of the *exec* family of functions. The following is a description of the actions taken:

1. If the process image file is a valid executable (in a format that is executable and valid and having appropriate privileges) for this system, then the system executes the file.
2. If the process image file has appropriate privileges and is in a format that is executable but not valid for this system (such as a recognized binary for another architecture), then this is an error and *errno* is set to [EINVAL] (see later RATIONALE on [EINVAL]).
3. If the process image file has appropriate privileges but is not otherwise recognized:
 - a. If this is a call to *execlp()* or *execvp()*, then they invoke a command interpreter assuming that the process image file is a shell script.
 - b. If this is not a call to *execlp()* or *execvp()*, then an error occurs and *errno* is set to [ENOEXEC].

Applications that do not require to access their arguments may use the form:

```
main(void)
```

as specified in the ISO C standard. However, the implementation will always provide the two

arguments *argc* and *argv*, even if they are not used.

Some implementations provide a third argument to *main()* called *envp*. This is defined as a pointer to the environment. The ISO C standard specifies invoking *main()* with two arguments, so implementations must support applications written this way. Since this volume of POSIX.1-2008 defines the global variable *environ*, which is also provided by historical implementations and can be used anywhere that *envp* could be used, there is no functional need for the *envp* argument. Applications should use the *getenv()* function rather than accessing the environment directly via either *envp* or *environ*. Implementations are required to support the two-argument calling sequence, but this does not prohibit an implementation from supporting *envp* as an optional third argument.

This volume of POSIX.1-2008 specifies that signals set to SIG_IGN remain set to SIG_IGN, and that the new process image inherits the signal mask of the thread that called *exec* in the old process image. This is consistent with historical implementations, and it permits some useful functionality, such as the *nohup* command. However, it should be noted that many existing applications wrongly assume that they start with certain signals set to the default action and/or unblocked. In particular, applications written with a simpler signal model that does not include blocking of signals, such as the one in the ISO C standard, may not behave properly if invoked with some signals blocked. Therefore, it is best not to block or ignore signals across *execs* without explicit reason to do so, and especially not to block signals across *execs* of arbitrary (not closely co-operating) programs.

The *exec* functions always save the value of the effective user ID and effective group ID of the process at the completion of the *exec*, whether or not the set-user-ID or the set-group-ID bit of the process image file is set.

The statement about *argv[]* and *envp[]* being constants is included to make explicit to future writers of language bindings that these objects are completely constant. Due to a limitation of the ISO C standard, it is not possible to state that idea in standard C. Specifying two levels of *const-qualification* for the *argv[]* and *envp[]* parameters for the *exec* functions may seem to be the natural choice, given that these functions do not modify either the array of pointers or the characters to which the function points, but this would disallow existing correct code. Instead, only the array of pointers is noted as constant. The table of assignment compatibility for *dst=src* derived from the ISO C standard summarizes the compatibility:

<i>dst:</i>	char *[]	const char *[]	char *const[]	const char *const[]
<i>src:</i>				
char *[]	VALID	—	VALID	—
const char *[]	—	VALID	—	VALID
char *const[]	—	—	VALID	—
const char *const[]	—	—	—	VALID

Since all existing code has a source type matching the first row, the column that gives the most valid combinations is the third column. The only other possibility is the fourth column, but using it would require a cast on the *argv* or *envp* arguments. It is unfortunate that the fourth column cannot be used, because the declaration a non-expert would naturally use would be that in the second row.

The ISO C standard and this volume of POSIX.1-2008 do not conflict on the use of *environ*, but some historical implementations of *environ* may cause a conflict. As long as *environ* is treated in the same way as an entry point (for example, *fork()*), it conforms to both standards. A library can contain *fork()*, but if there is a user-provided *fork()*, that *fork()* is given precedence and no problem ensues. The situation is similar for *environ*: the definition in this volume of POSIX.1-2008 is to be used if there is no user-provided *environ* to take precedence. At least three

26101	implementations are known to exist that solve this problem.	
26102	[E2BIG]	The limit {ARG_MAX} applies not just to the size of the argument list, but to the sum of that and the size of the environment list.
26103		
26104	[EFAULT]	Some historical systems return [EFAULT] rather than [ENOEXEC] when the new process image file is corrupted. They are non-conforming.
26105		
26106	[EINVAL]	This error condition was added to POSIX.1-2008 to allow an implementation to detect executable files generated for different architectures, and indicate this situation to the application. Historical implementations of shells, <i>execvp()</i> , and <i>execlp()</i> that encounter an [ENOEXEC] error will execute a shell on the assumption that the file is a shell script. This will not produce the desired effect when the file is a valid executable for a different architecture. An implementation may now choose to avoid this problem by returning [EINVAL] when a valid executable for a different architecture is encountered. Some historical implementations return [EINVAL] to indicate that the <i>path</i> argument contains a character with the high order bit set. The standard developers chose to deviate from historical practice for the following reasons:
26107		
26108		
26109		
26110		
26111		
26112		
26113		
26114		
26115		
26116		
26117		1. The new utilization of [EINVAL] will provide some measure of utility to the user community.
26118		
26119		2. Historical use of [EINVAL] is not acceptable in an internationalized operating environment.
26120		
26121	[ENAMETOOLONG]	
26122		Since the file pathname may be constructed by taking elements in the <i>PATH</i> variable and putting them together with the filename, the [ENAMETOOLONG] error condition could also be reached this way.
26123		
26124		
26125	[ETXTBSY]	System V returns this error when the executable file is currently open for writing by some process. This volume of POSIX.1-2008 neither requires nor prohibits this behavior.
26126		
26127		
26128	Other systems (such as System V) may return [EINTR] from <i>exec</i> . This is not addressed by this volume of POSIX.1-2008, but implementations may have a window between the call to <i>exec</i> and the time that a signal could cause one of the <i>exec</i> calls to return with [EINTR].	
26129		
26130		
26131	An explicit statement regarding the floating-point environment (as defined in the <i><fenv.h></i> header) was added to make it clear that the floating-point environment is set to its default when a call to one of the <i>exec</i> functions succeeds. The requirements for inheritance or setting to the default for other process and thread start-up functions is covered by more generic statements in their descriptions and can be summarized as follows:	
26132		
26133		
26134		
26135		
26136	<i>posix_spawn()</i>	Set to default.
26137	<i>fork()</i>	Inherit.
26138	<i>pthread_create()</i>	Inherit.
26139	The purpose of the <i>fexecve()</i> function is to enable executing a file which has been verified to be the intended file. It is possible to actively check the file by reading from the file descriptor and be sure that the file is not exchanged for another between the reading and the execution. Alternatively, an function like <i>openat()</i> can be used to open a file which has been found by reading the content of a directory using <i>readdir()</i> .	
26140		
26141		
26142		
26143		

FUTURE DIRECTIONS

None.

SEE ALSO

alarm(), *atexit()*, *chmod()*, *close()*, *confstr()*, *exit()*, *fcntl()*, *fork()*, *fstatvfs()*, *getenv()*, *getitimer()*, *getrlimit()*, *mknod()*, *mmap()*, *nice()*, *open()*, *posix_spawn()*, *posix_trace_create()*, *posix_trace_event()*, *posix_trace_eventid_equal()*, *pthread_atfork()*, *pthread_sigmask()*, *putenv()*, *readdir()*, *semop()*, *setlocale()*, *shmat()*, *sigaction()*, *sigaltstack()*, *sigpending()*, *system()*, *times()*, *ulimit()*, *umask()*

XBD Chapter 8 (on page 173), [`<unistd.h>`](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, behavior is defined for when the process image file is not a valid executable.
- In this version, `_POSIX_SAVED_IDS` is mandated, thus the effective user ID and effective group ID of the new process image shall be saved (as the saved set-user-ID and the saved set-group-ID) for use by the `setuid()` function.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [EINVAL] mandatory error condition is added.
- The [ELOOP] optional error condition is added.

The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for typed memory.

The normative text is updated to avoid use of the term “must” for application requirements.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

IEEE PASC Interpretation 1003.1 #132 is applied.

The DESCRIPTION is updated to make it explicit that the floating-point environment in the new process image is set to the default.

The DESCRIPTION and RATIONALE are updated to include clarifications of how the contents of a process image file affect the behavior of the *exec* functions.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/15 is applied, adding a new paragraph to the DESCRIPTION and text to the end of the APPLICATION USAGE section. This change

exec*System Interfaces*

- 26185 addresses a security concern, where implementations may want to reopen file descriptors 0, 1,
 26186 and 2 for programs with the set-user-id or set-group-id file mode bits calling the *exec* family of
 26187 functions.
- 26188 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/24 is applied, applying changes to the
 26189 DESCRIPTION, addressing which attributes are inherited by threads, and behavioral
 26190 requirements for threads attributes.
- 26191 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/25 is applied, updating text in the
 26192 RATIONALE from “the process signal mask be unchanged across an *exec*” to “the new process
 26193 image inherits the signal mask of the thread that called *exec* in the old process image”.
- 26194 **Issue 7**
- 26195 Austin Group Interpretation 1003.1-2001 #047 is applied, adding the description of `_CS_V7_ENV`
 26196 to the APPLICATION USAGE.
- 26197 Austin Group Interpretation 1003.1-2001 #143 is applied.
- 26198 The *fexecve()* function is added from The Open Group Technical Standard, 2006, Extended API
 26199 Set Part 2.
- 26200 Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Threads,
 26201 and Timers options is moved to the Base.
- 26202 Changes are made related to support for finegrained timestamps.

System Interfaces

exit()26203 **NAME**26204 `exit` — terminate a process26205 **SYNOPSIS**26206 `#include <stdlib.h>`26207 `void exit(int status);`26208 **DESCRIPTION**

26209 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 26210 conflict between the requirements described here and the ISO C standard is unintentional. This
 26211 volume of POSIX.1-2008 defers to the ISO C standard.

26212 CX The value of *status* may be 0, EXIT_SUCCESS, EXIT_FAILURE, or any other value, though only
 26213 the least significant 8 bits (that is, *status* & 0377) shall be available to a waiting parent process.

26214 The *exit()* function shall first call all functions registered by *atexit()*, in the reverse order of their
 26215 registration, except that a function is called after any previously registered functions that had
 26216 already been called at the time it was registered. Each function is called as many times as it was
 26217 registered. If, during the call to any such function, a call to the *longjmp()* function is made that
 26218 would terminate the call to the registered function, the behavior is undefined.

26219 If a function registered by a call to *atexit()* fails to return, the remaining registered functions shall
 26220 not be called and the rest of the *exit()* processing shall not be completed. If *exit()* is called more
 26221 than once, the behavior is undefined.

26222 The *exit()* function shall then flush all open streams with unwritten buffered data and close all
 26223 CX open streams. Finally, the process shall be terminated with the same consequences as described
 26224 in [Consequences of Process Termination](#) (on page 545).

26225 **RETURN VALUE**26226 The *exit()* function does not return.26227 **ERRORS**

26228 No errors are defined.

26229 **EXAMPLES**

26230 None.

26231 **APPLICATION USAGE**

26232 None.

26233 **RATIONALE**26234 See *_Exit()*.26235 **FUTURE DIRECTIONS**

26236 None.

26237 **SEE ALSO**26238 [_Exit\(\)](#), [atexit\(\)](#), [exec](#), [longjmp\(\)](#), [tmpfile\(\)](#)26239 XBD [<stdlib.h>](#)26240 **CHANGE HISTORY**26241 **Issue 7**

26242 Austin Group Interpretation 1003.1-2001 #031 is applied, separating the *_Exit()* and *_exit()*
 26243 functions from the *exit()* function.

26244 Austin Group Interpretation 1003.1-2001 #085 is applied.

exp()26245 **NAME**26246 `exp, expf, expl` — exponential function26247 **SYNOPSIS**26248 `#include <math.h>`26249 `double exp(double x);`26250 `float expf(float x);`26251 `long double expl(long double x);`26252 **DESCRIPTION**

26253 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 26254 conflict between the requirements described here and the ISO C standard is unintentional. This
 26255 volume of POSIX.1-2008 defers to the ISO C standard.

26256 These functions shall compute the base-*e* exponential of *x*.

26257 An application wishing to check for error situations should set *errno* to zero and call
 26258 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 26259 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 26260 zero, an error has occurred.

26261 **RETURN VALUE**26262 Upon successful completion, these functions shall return the exponential value of *x*.

26263 If the correct value would cause overflow, a range error shall occur and *exp()*, *expf()*, and *expl()*
 26264 shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

26265 If the correct value would cause underflow, and is not representable, a range error may occur,
 26266 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

26267 MX If *x* is NaN, a NaN shall be returned.26268 If *x* is ± 0 , 1 shall be returned.26269 If *x* is $-\text{Inf}$, +0 shall be returned.26270 If *x* is $+\text{Inf}$, *x* shall be returned.

26271 If the correct value would cause underflow, and is representable, a range error may occur and
 26272 the correct value shall be returned.

26273 **ERRORS**

26274 These functions shall fail if:

26275 Range Error The result overflows.

26276 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 26277 then *errno* shall be set to [ERANGE]. If the integer expression
 26278 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 26279 floating-point exception shall be raised.

26280 These functions may fail if:

26281 Range Error The result underflows.

26282 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 26283 then *errno* shall be set to [ERANGE]. If the integer expression
 26284 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 26285 floating-point exception shall be raised.

26286 **EXAMPLES**26287 **Computing the Density of the Standard Normal Distribution**

26288 This function shows an implementation for the density of the standard normal distribution
 26289 using *exp()*. This example uses the constant `M_PI` which is part of the XSI option.

```
26290 #include <math.h>
26291
26292 double
26293 normal_density (double x)
26294 {
26295     return exp(-x*x/2) / sqrt (2*M_PI);
26296 }
```

26296 **APPLICATION USAGE**

26297 Note that for IEEE Std 754-1985 **double**, $709.8 < x$ implies *exp(x)* has overflowed. The value
 26298 $x < -708.4$ implies *exp(x)* has underflowed.

26299 On error, the expressions (*math_errhandling* & `MATH_ERRNO`) and (*math_errhandling* &
 26300 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

26301 **RATIONALE**

26302 None.

26303 **FUTURE DIRECTIONS**

26304 None.

26305 **SEE ALSO**

26306 *feclearexcept()*, *fetestexcept()*, *isnan()*, *log()*

26307 XBD Section 4.19 (on page 116), `<math.h>`

26308 **CHANGE HISTORY**

26309 First released in Issue 1. Derived from Issue 1 of the SVID.

26310 **Issue 5**

26311 The DESCRIPTION is updated to indicate how an application should check for an error. This
 26312 text was previously published in the APPLICATION USAGE section.

26313 **Issue 6**

26314 The *expf()* and *expl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

26315 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 26316 revised to align with the ISO/IEC 9899:1999 standard.

26317 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 26318 marked.

26319 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/26 is applied, adding the example to the
 26320 EXAMPLES section.

exp2()26321 **NAME**

26322 exp2, exp2f, exp2l — exponential base 2 functions

26323 **SYNOPSIS**

```
26324 #include <math.h>
26325
26325 double exp2(double x);
26326 float exp2f(float x);
26327 long double exp2l(long double x);
```

26328 **DESCRIPTION**

26329 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 26330 conflict between the requirements described here and the ISO C standard is unintentional. This
 26331 volume of POSIX.1-2008 defers to the ISO C standard.

26332 These functions shall compute the base-2 exponential of x .

26333 An application wishing to check for error situations should set *errno* to zero and call
 26334 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 26335 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 26336 zero, an error has occurred.

26337 **RETURN VALUE**

26338 Upon successful completion, these functions shall return 2^x .

26339 If the correct value would cause overflow, a range error shall occur and *exp2()*, *exp2f()*, and
 26340 *exp2l()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 26341 respectively.

26342 If the correct value would cause underflow, and is not representable, a range error may occur,
 26343 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

26344 MX If x is NaN, a NaN shall be returned.

26345 If x is ± 0 , 1 shall be returned.

26346 If x is $-\text{Inf}$, +0 shall be returned.

26347 If x is $+\text{Inf}$, x shall be returned.

26348 If the correct value would cause underflow, and is representable, a range error may occur and
 26349 the correct value shall be returned.

26350 **ERRORS**

26351 These functions shall fail if:

26352 Range Error The result overflows.

26353 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 26354 then *errno* shall be set to [ERANGE]. If the integer expression
 26355 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 26356 floating-point exception shall be raised.

26357 These functions may fail if:

26358 Range Error The result underflows.

26359 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 26360 then *errno* shall be set to [ERANGE]. If the integer expression
 26361 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 26362 floating-point exception shall be raised.

26363 **EXAMPLES**

26364 None.

26365 **APPLICATION USAGE**

26366 For IEEE Std 754-1985 **double**, $1024 \leq x$ implies $\exp2(x)$ has overflowed. The value $x < -1022$
 26367 implies $\exp(x)$ has underflowed.

26368 On error, the expressions $(\text{math\_errhandling} \ \& \ \text{MATH\_ERRNO})$ and $(\text{math\_errhandling} \ \& \ \text{MATH\_ERREXCEPT})$ are independent of each other, but at least one of them must be non-zero.

26370 **RATIONALE**

26371 None.

26372 **FUTURE DIRECTIONS**

26373 None.

26374 **SEE ALSO**26375 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *log()*26376 XBD Section 4.19 (on page 116), **<math.h>**26377 **CHANGE HISTORY**

26378 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

expm1()*System Interfaces*26379 **NAME**

26380 expm1, expm1f, expm1l — compute exponential functions

26381 **SYNOPSIS**

```

26382       #include <math.h>

26383       double expm1(double x);
26384       float expm1f(float x);
26385       long double expm1l(long double x);

```

26386 **DESCRIPTION**

26387 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 26388 conflict between the requirements described here and the ISO C standard is unintentional. This
 26389 volume of POSIX.1-2008 defers to the ISO C standard.

26390 These functions shall compute $e^x - 1.0$.

26391 An application wishing to check for error situations should set *errno* to zero and call
 26392 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 26393 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 26394 zero, an error has occurred.

26395 **RETURN VALUE**

26396 Upon successful completion, these functions return $e^x - 1.0$.

26397 If the correct value would cause overflow, a range error shall occur and *expm1()*, *expm1f()*, and
 26398 *expm1l()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 26399 respectively.

26400 **MX** If *x* is NaN, a NaN shall be returned.
 26401 If *x* is ± 0 , ± 0 shall be returned.
 26402 If *x* is $-\text{Inf}$, -1 shall be returned.
 26403 If *x* is $+\text{Inf}$, *x* shall be returned.
 26404 If *x* is subnormal, a range error may occur and *x* should be returned.

26405 **ERRORS**

26406 These functions shall fail if:

26407 Range Error The result overflows.

26408 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 26409 then *errno* shall be set to [ERANGE]. If the integer expression
 26410 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 26411 floating-point exception shall be raised.

26412 These functions may fail if:

26413 **MX** Range Error The value of *x* is subnormal.

26414 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 26415 then *errno* shall be set to [ERANGE]. If the integer expression
 26416 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 26417 floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

The value of $\text{expm1}(x)$ may be more accurate than $\exp(x)-1.0$ for small values of x .

The $\text{expm1}()$ and $\log1p()$ functions are useful for financial calculations of $((1+x)^n-1)/x$, namely:

$$\text{expm1}(n * \log1p(x)) / x$$

when x is very small (for example, when calculating small daily interest rates). These functions also simplify writing accurate inverse hyperbolic functions.

For IEEE Std 754-1985 **double**, $709.8 < x$ implies $\text{expm1}(x)$ has overflowed.

On error, the expressions $(\text{math\_errhandling} \ \& \ \text{MATH\_ERRNO})$ and $(\text{math\_errhandling} \ \& \ \text{MATH\_ERREXCEPT})$ are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[*exp\(\)*](#), [*feclearexcept\(\)*](#), [*fetestexcept\(\)*](#), [*ilogb\(\)*](#), [*log1p\(\)*](#)

XBD [Section 4.19](#) (on page 116), [**<math.h>**](#)

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The $\text{expm1f}()$ and $\text{expm1l}()$ functions are added for alignment with the ISO/IEC 9899:1999 standard.

The $\text{expm1}()$ function is no longer marked as an extension.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

fabs()*System Interfaces*26448 **NAME**26449 **fabs, fabsf, fabsl** — absolute value function26450 **SYNOPSIS**

```

26451     #include <math.h>

26452     double fabs(double x);
26453     float fabsf(float x);
26454     long double fabsl(long double x);

```

26455 **DESCRIPTION**

26456 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 26457 conflict between the requirements described here and the ISO C standard is unintentional. This
 26458 volume of POSIX.1-2008 defers to the ISO C standard.

26459 These functions shall compute the absolute value of their argument x , $|x|$.

26460 **RETURN VALUE**

26461 Upon successful completion, these functions shall return the absolute value of x .

26462 MX If x is NaN, a NaN shall be returned.

26463 If x is ± 0 , $+0$ shall be returned.

26464 If x is $\pm \text{Inf}$, $+\text{Inf}$ shall be returned.

26465 **ERRORS**

26466 No errors are defined.

26467 **EXAMPLES**26468 **Computing the 1-Norm of a Floating-Point Vector**

26469 This example shows the use of *fabs()* to compute the 1-norm of a vector defined as follows:

```

26470 norm1(v) = |v[0]| + |v[1]| + ... + |v[n-1]|

```

26471 where $|x|$ denotes the absolute value of x , n denotes the vector's dimension $v[i]$ denotes the i -th
 26472 component of v ($0 \leq i < n$).

```

26473 #include <math.h>

26474 double
26475 norm1(const double v[], const int n)
26476 {
26477     int i;
26478     double n1_v; /* 1-norm of v */
26479     n1_v = 0;
26480     for (i=0; i<n; i++) {
26481         n1_v += fabs (v[i]);
26482     }
26483     return n1_v;
26484 }

```

26485 **APPLICATION USAGE**

26486 None.

26487 **RATIONALE**

26488 None.

26489 **FUTURE DIRECTIONS**

26490 None.

26491 **SEE ALSO**26492 *isnan()*26493 XBD *<math.h>*26494 **CHANGE HISTORY**

26495 First released in Issue 1. Derived from Issue 1 of the SVID.

26496 **Issue 5**

26497 The DESCRIPTION is updated to indicate how an application should check for an error. This
 26498 text was previously published in the APPLICATION USAGE section.

26499 **Issue 6**

26500 The *fabsf()* and *fabsl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

26501 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 26502 revised to align with the ISO/IEC 9899:1999 standard.

26503 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 26504 marked.

26505 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/27 is applied, adding the example to the
 26506 EXAMPLES section.

faccessat()*System Interfaces*26507 **NAME**26508 **faccessat** — determine accessibility of a file relative to directory file descriptor26509 **SYNOPSIS**26510 `#include <unistd.h>`26511 `int faccessat(int fd, const char *path, int amode, int flag);`26512 **DESCRIPTION**26513 Refer to *access()*.

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

NAME

fattach — attach a STREAMS-based file descriptor to a file in the file system name space (**STREAMS**)

SYNOPSIS

```
#include <stropts.h>

int fattach(int fildes, const char *path);
```

DESCRIPTION

The *fattach()* function shall attach a STREAMS-based file descriptor to a file, effectively associating a pathname with *fildes*. The application shall ensure that the *fildes* argument is a valid open file descriptor associated with a STREAMS file. The *path* argument points to a pathname of an existing file. The application shall have appropriate privileges or be the owner of the file named by *path* and have write permission. A successful call to *fattach()* shall cause all pathnames that name the file named by *path* to name the STREAMS file associated with *fildes*, until the STREAMS file is detached from the file. A STREAMS file can be attached to more than one file and can have several pathnames associated with it.

The attributes of the named STREAMS file shall be initialized as follows: the permissions, user ID, group ID, and times are set to those of the file named by *path*, the number of links is set to 1, and the size and device identifier are set to those of the STREAMS file associated with *fildes*. If any attributes of the named STREAMS file are subsequently changed (for example, by *chmod()*), neither the attributes of the underlying file nor the attributes of the STREAMS file to which *fildes* refers shall be affected.

File descriptors referring to the underlying file, opened prior to an *fattach()* call, shall continue to refer to the underlying file.

RETURN VALUE

Upon successful completion, *fattach()* shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *fattach()* function shall fail if:

[EACCES]	Search permission is denied for a component of the path prefix, or the process is the owner of <i>path</i> but does not have write permissions on the file named by <i>path</i> .
[EBADF]	The <i>fildes</i> argument is not a valid open file descriptor.
[EBUSY]	The file named by <i>path</i> is currently a mount point or has a STREAMS file attached to it.
[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
[ENAMETOOLONG]	The length of a component of a pathname is longer than {NAME_MAX}.
[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
[ENOTDIR]	A component of the path prefix is not a directory, or the <i>path</i> argument contains at least one non- <i><slash></i> character and ends with one or more trailing <i><slash></i> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

fattach()*System Interfaces*

26557	[EPERM]	The effective user ID of the process is not the owner of the file named by <i>path</i> and the process does not have appropriate privileges.
26558		
26559		The <i>fattach()</i> function may fail if:
26560	[EINVAL]	The <i>files</i> argument does not refer to a STREAMS file.
26561	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
26562		
26563	[ENAMETOOLONG]	
26564		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
26565		
26566		
26567	[EXDEV]	A link to a file on another file system was attempted.

26568 **EXAMPLES**26569 **Attaching a File Descriptor to a File**

26570 In the following example, *fd* refers to an open STREAMS file. The call to *fattach()* associates this
 26571 STREAM with the file **/tmp/named-STREAM**, such that any future calls to open **/tmp/named-
 26572 STREAM**, prior to breaking the attachment via a call to *fdetach()*, will instead create a new file
 26573 handle referring to the STREAMS file associated with *fd*.

```
26574 #include <stropts.h>
26575 ...
26576     int fd;
26577     char *filename = "/tmp/named-STREAM";
26578     int ret;
26579
26579     ret = fattach(fd, filename);
```

26580 **APPLICATION USAGE**

26581 The *fattach()* function behaves similarly to the traditional *mount()* function in the way a file is
 26582 temporarily replaced by the root directory of the mounted file system. In the case of *fattach()*, the
 26583 replaced file need not be a directory and the replacing file is a STREAMS file.

26584 **RATIONALE**

26585 The file attributes of a file which has been the subject of an *fattach()* call are specifically set
 26586 because of an artifact of the original implementation. The internal mechanism was the same as
 26587 for the *mount()* function. Since *mount()* is typically only applied to directories, the effects when
 26588 applied to a regular file are a little surprising, especially as regards the link count which rigidly
 26589 remains one, even if there were several links originally and despite the fact that all original links
 26590 refer to the STREAM as long as the *fattach()* remains in effect.

26591 **FUTURE DIRECTIONS**

26592 The *fattach()* function may be removed in a future version.

26593 **SEE ALSO**

26594 *fdetach()*, *isastream()*

26595 XBD **<stropts.h>**

26596 **CHANGE HISTORY**

26597 First released in Issue 4, Version 2.

Issue 5

26598 Moved from X/OPEN UNIX extension to BASE.

26600 The [EXDEV] error is added to the list of optional errors in the ERRORS section.

Issue 6

26601 This function is marked as part of the XSI STREAMS Option Group.

26603 The normative text is updated to avoid use of the term “must” for application requirements.

26604 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
26605 [ELOOP] error condition is added.

Issue 7

26606 Austin Group Interpretation 1003.1-2001 #143 is applied.

26607 The *fattach()* function is marked obsolescent.

26609 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
26610 pathname exists but is not a directory or a symbolic link to a directory.

fchdir()*System Interfaces*26611 **NAME**26612 **fchdir** — change working directory26613 **SYNOPSIS**26614 `#include <unistd.h>`26615 `int fchdir(int fildes);`26616 **DESCRIPTION**26617 The *fchdir()* function shall be equivalent to *chdir()* except that the directory that is to be the new
26618 current working directory is specified by the file descriptor *fildes*.26619 A conforming application can obtain a file descriptor for a file of type directory using *open()*,
26620 provided that the file status flags and access modes do not contain *O_WRONLY* or *O_RDWR*.26621 **RETURN VALUE**26622 Upon successful completion, *fchdir()* shall return 0. Otherwise, it shall return *-1* and set *errno* to
26623 indicate the error. On failure the current working directory shall remain unchanged.26624 **ERRORS**26625 The *fchdir()* function shall fail if:26626 [*EACCES*] Search permission is denied for the directory referenced by *fildes*.26627 [*EBADF*] The *fildes* argument is not an open file descriptor.26628 [*ENOTDIR*] The open file descriptor *fildes* does not refer to a directory.26629 The *fchdir()* may fail if:26630 [*EINTR*] A signal was caught during the execution of *fchdir()*.26631 [*EIO*] An I/O error occurred while reading from or writing to the file system.26632 **EXAMPLES**

26633 None.

26634 **APPLICATION USAGE**

26635 None.

26636 **RATIONALE**

26637 None.

26638 **FUTURE DIRECTIONS**

26639 None.

26640 **SEE ALSO**26641 *chdir()*, *dirfd()*26642 *XBD* <unistd.h>26643 **CHANGE HISTORY**

26644 First released in Issue 4, Version 2.

26645 **Issue 5**

26646 Moved from X/OPEN UNIX extension to BASE.

26647 **Issue 7**26648 The *fchdir()* function is moved from the XSI option to the Base.

NAME

fchmod — change mode of a file

SYNOPSIS

```
#include <sys/stat.h>

int fchmod(int fildevs, mode_t mode);
```

DESCRIPTION

The *fchmod()* function shall be equivalent to *chmod()* except that the file whose permissions are changed is specified by the file descriptor *fildevs*.

SHM

If *fildevs* references a shared memory object, the *fchmod()* function need only affect the *S_IRUSR*, *S_IWUSR*, *S_IRGRP*, *S_IWGRP*, *S_IROTH*, and *S_IWOTH* file permission bits.

TYM

If *fildevs* references a typed memory object, the behavior of *fchmod()* is unspecified.

If *fildevs* refers to a socket, the behavior of *fchmod()* is unspecified.

OB XSR

If *fildevs* refers to a STREAM (which is *fattach()*-ed into the file system name space) the call returns successfully, doing nothing.

RETURN VALUE

Upon successful completion, *fchmod()* shall return 0. Otherwise, it shall return *-1* and set *errno* to indicate the error.

ERRORS

The *fchmod()* function shall fail if:

[EBADF] The *fildevs* argument is not an open file descriptor.

[EPERM] The effective user ID does not match the owner of the file and the process does not have appropriate privileges.

[EROFS] The file referred to by *fildevs* resides on a read-only file system.

The *fchmod()* function may fail if:

[EINTR] The *fchmod()* function was interrupted by a signal.

[EINVAL] The value of the *mode* argument is invalid.

[EINVAL] The *fildevs* argument refers to a pipe and the implementation disallows execution of *fchmod()* on a pipe.

EXAMPLES**Changing the Current Permissions for a File**

The following example shows how to change the permissions for a file named */home/cnd/mod1* so that the owner and group have read/write/execute permissions, but the world only has read/write permissions.

```
#include <sys/stat.h>
#include <fcntl.h>

mode_t mode;
int fildevs;
...
fildevs = open("/home/cnd/mod1", O_RDWR);
fchmod(fildevs, S_IRWXU | S_IRWXG | S_IROTH | S_IWOTH);
```

fchmod()*System Interfaces*26689 **APPLICATION USAGE**

26690 None.

26691 **RATIONALE**

26692 None.

26693 **FUTURE DIRECTIONS**

26694 None.

26695 **SEE ALSO**26696 *chmod(), chown(), creat(), fcntl(), fstatat(), fstatvfs(), mknod(), open(), read(), write()*26697 XBD [<sys/stat.h>](#)26698 **CHANGE HISTORY**

26699 First released in Issue 4, Version 2.

26700 **Issue 5**

26701 Moved from X/OPEN UNIX extension to BASE and aligned with *fchmod()* in the POSIX
 26702 Realtime Extension. Specifically, the second paragraph of the DESCRIPTION is added and a
 26703 second instance of [EINVAL] is defined in the list of optional errors.

26704 **Issue 6**

26705 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by stating that *fchmod()*
 26706 behavior is unspecified for typed memory objects.

*System Interfaces***fchmodat()**26707 **NAME**

26708 fchmodat — change mode of a file relative to directory file descriptor

26709 **SYNOPSIS**

26710 #include <sys/stat.h>

26711 int fchmodat(int *fd*, const char **path*, mode_t *mode*, int *flag*);26712 **DESCRIPTION**26713 Refer to *chmod()*.

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

fchown()

System Interfaces

26714 **NAME**26715 **fchown** — change owner and group of a file26716 **SYNOPSIS**26717 `#include <unistd.h>`26718 `int fchown(int fildes, uid_t owner, gid_t group);`26719 **DESCRIPTION**

26720 The *fchown()* function shall be equivalent to *chown()* except that the file whose owner and group
 26721 are changed is specified by the file descriptor *fil*des.

26722 **RETURN VALUE**

26723 Upon successful completion, *fchown()* shall return 0. Otherwise, it shall return `-1` and set *errno* to
 26724 indicate the error.

26725 **ERRORS**26726 The *fchown()* function shall fail if:26727 [EBADF] The *fil*des argument is not an open file descriptor.

26728 [EPERM] The effective user ID does not match the owner of the file or the process does
 26729 not have appropriate privileges and `_POSIX_CHOWN_RESTRICTED`
 26730 indicates that such privilege is required.

26731 [EROFS] The file referred to by *fil*des resides on a read-only file system.26732 The *fchown()* function may fail if:

26733 [EINVAL] The owner or group ID is not a value supported by the implementation. The
 26734 *fil*des argument refers to a pipe or socket or an *fattach()*-ed STREAM and the
 26735 implementation disallows execution of *fchown()* on a pipe.

26736 [EIO] A physical I/O error has occurred.

26737 [EINTR] The *fchown()* function was interrupted by a signal which was caught.26738 **EXAMPLES**26739 **Changing the Current Owner of a File**

26740 The following example shows how to change the owner of a file named `/home/cnd/mod1` to
 26741 “jones” and the group to “cnd”.

26742 The numeric value for the user ID is obtained by extracting the user ID from the user database
 26743 entry associated with “jones”. Similarly, the numeric value for the group ID is obtained by
 26744 extracting the group ID from the group database entry associated with “cnd”. This example
 26745 assumes the calling program has appropriate privileges.

26746 `#include <sys/types.h>`26747 `#include <unistd.h>`26748 `#include <fcntl.h>`26749 `#include <pwd.h>`26750 `#include <grp.h>`26751 `struct passwd *pwd;`26752 `struct group *grp;`26753 `int fil`des;26754 `...`26755 `fil`des = `open("/home/cnd/mod1", O_RDWR);`26756 `pwd = getpwnam("jones");`

```

26757     grp = getgrnam("cnd");
26758     fchown(fildes, pwd->pw_uid, grp->gr_gid);

```

26759 APPLICATION USAGE

26760 None.

26761 RATIONALE

26762 None.

26763 FUTURE DIRECTIONS

26764 None.

26765 SEE ALSO

26766 *chown()*

26767 XBD *<unistd.h>*

26768 CHANGE HISTORY

26769 First released in Issue 4, Version 2.

26770 Issue 5

26771 Moved from X/OPEN UNIX extension to BASE.

26772 Issue 6

26773 The following changes were made to align with the IEEE P1003.1a draft standard:

- 26774 • Clarification is added that a call to *fchown()* may not be allowed on a pipe.

26775 The *fchown()* function is defined as mandatory.

26776 Issue 7

26777 Functionality relating to XSI STREAMS is marked obsolescent.

fchownat()*System Interfaces*26778 **NAME**

26779 fchownat — change owner and group of a file relative to directory file descriptor

26780 **SYNOPSIS**

26781 #include <unistd.h>

26782 int fchownat(int *fd*, const char **path*, uid_t *owner*, gid_t *group*,26783 int *flag*);26784 **DESCRIPTION**26785 Refer to *chown()*.

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

26786 **NAME**26787 `fclose` — close a stream26788 **SYNOPSIS**26789 `#include <stdio.h>`26790 `int fclose(FILE *stream);`26791 **DESCRIPTION**

26792 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 26793 conflict between the requirements described here and the ISO C standard is unintentional. This
 26794 volume of POSIX.1-2008 defers to the ISO C standard.

26795 The `fclose()` function shall cause the stream pointed to by *stream* to be flushed and the associated
 26796 file to be closed. Any unwritten buffered data for the stream shall be written to the file; any
 26797 unread buffered data shall be discarded. Whether or not the call succeeds, the stream shall be
 26798 disassociated from the file and any buffer set by the `setbuf()` or `setvbuf()` function shall be
 26799 disassociated from the stream. If the associated buffer was automatically allocated, it shall be
 26800 deallocated.

26801 CX If the file is not already at EOF, and the file is one capable of seeking, the file offset of the
 26802 underlying open file description shall be adjusted so that the next operation on the open file
 26803 description deals with the byte after the last one read from or written to the stream being closed.

26804 The `fclose()` function shall mark for update the last data modification and last file status change
 26805 timestamps of the underlying file, if the stream was writable, and if buffered data remains that
 26806 has not yet been written to the file. The `fclose()` function shall perform the equivalent of a `close()`
 26807 on the file descriptor that is associated with the stream pointed to by *stream*.

26808 After the call to `fclose()`, any use of *stream* results in undefined behavior.

26809 **RETURN VALUE**

26810 CX Upon successful completion, `fclose()` shall return 0; otherwise, it shall return EOF and set *errno*
 26811 to indicate the error.

26812 **ERRORS**

26813 The `fclose()` function shall fail if:

26814 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and
 26815 the thread would be delayed in the write operation.

26816 CX [EBADF] The file descriptor underlying stream is not valid.

26817 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

26818 XSI [EFBIG] An attempt was made to write a file that exceeds the file size limit of the
 26819 process.

26820 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 26821 offset maximum associated with the corresponding stream.

26822 CX [EINTR] The `fclose()` function was interrupted by a signal.

26823 CX [EIO] The process is a member of a background process group attempting to write to
 26824 its controlling terminal, TOSTOP is set, the process is neither ignoring nor
 26825 blocking SIGTTOU, and the process group of the process is orphaned. This
 26826 error may also be returned under implementation-defined conditions.

26827 CX [ENOMEM] The underlying stream was created by `open_memstream()` or
 26828 `open_wmemstream()` and insufficient memory is available.

fclose()*System Interfaces*

26829 CX [ENOSPC] There was no free space remaining on the device containing the file or in the
26830 buffer used by the *fmemopen()* function.

26831 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
26832 any process. A SIGPIPE signal shall also be sent to the thread.

26833 The *fclose()* function may fail if:

26834 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
26835 capabilities of the device.

26836 **EXAMPLES**

26837 None.

26838 **APPLICATION USAGE**

26839 None.

26840 **RATIONALE**

26841 None.

26842 **FUTURE DIRECTIONS**

26843 None.

26844 **SEE ALSO**

26845 *close()*, *fmemopen()*, *fopen()*, *getrlimit()*, *open_memstream()*, *ulimit()*

26846 XBD <stdio.h>

26847 **CHANGE HISTORY**

26848 First released in Issue 1. Derived from Issue 1 of the SVID.

26849 **Issue 5**

26850 Large File Summit extensions are added.

26851 **Issue 6**

26852 Extensions beyond the ISO C standard are marked.

26853 The following new requirements on POSIX implementations derive from alignment with the
26854 Single UNIX Specification:

- 26855 • The [EFBIG] error is added as part of the large file support extensions.
- 26856 • The [ENXIO] optional error condition is added.

26857 The DESCRIPTION is updated to note that the stream and any buffer are disassociated whether
26858 or not the call succeeds. This is for alignment with the ISO/IEC 9899:1999 standard.

26859 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/28 is applied, updating the [EAGAIN]
26860 error in the ERRORS section from “the process would be delayed” to “the thread would be
26861 delayed”.

26862 **Issue 7**

26863 Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file
26864 descriptors and streams.

26865 The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open
26866 Group Technical Standard, 2006, Extended API Set Part 1.

26867 Changes are made related to support for finegrained timestamps.

NAME

fcntl — file control

SYNOPSIS

```
#include <fcntl.h>

int fcntl(int fildes, int cmd, ...);
```

DESCRIPTION

The *fcntl()* function shall perform the operations described below on open files. The *fil*des argument is a file descriptor.

The available values for *cmd* are defined in **<fcntl.h>** and are as follows:

F_DUPFD Return a new file descriptor which shall be the lowest numbered available (that is, not already open) file descriptor greater than or equal to the third argument, *arg*, taken as an integer of type **int**. The new file descriptor shall refer to the same open file description as the original file descriptor, and shall share any locks. The FD_CLOEXEC flag associated with the new file descriptor shall be cleared to keep the file open across calls to one of the *exec* functions.

F_DUPFD_CLOEXEC

Like F_DUPFD, but the FD_CLOEXEC flag associated with the new file descriptor shall be set.

F_GETFD

Get the file descriptor flags defined in **<fcntl.h>** that are associated with the file descriptor *fil*des. File descriptor flags are associated with a single file descriptor and do not affect other file descriptors that refer to the same file.

F_SETFD

Set the file descriptor flags defined in **<fcntl.h>**, that are associated with *fil*des, to the third argument, *arg*, taken as type **int**. If the FD_CLOEXEC flag in the third argument is 0, the file descriptor shall remain open across the *exec* functions; otherwise, the file descriptor shall be closed upon successful execution of one of the *exec* functions.

F_GETFL

Get the file status flags and file access modes, defined in **<fcntl.h>**, for the file description associated with *fil*des. The file access modes can be extracted from the return value using the mask O_ACCMODE, which is defined in **<fcntl.h>**. File status flags and file access modes are associated with the file description and do not affect other file descriptors that refer to the same file with different open file descriptions. The flags returned may include non-standard file status flags which the application did not set, provided that these additional flags do not alter the behavior of a conforming application.

F_SETFL

Set the file status flags, defined in **<fcntl.h>**, for the file description associated with *fil*des from the corresponding bits in the third argument, *arg*, taken as type **int**. Bits corresponding to the file access mode and the file creation flags, as defined in **<fcntl.h>**, that are set in *arg* shall be ignored. If any bits in *arg* other than those mentioned here are changed by the application, the result is unspecified.

F_GETOWN

If *fil*des refers to a socket, get the process or process group ID specified to receive SIGURG signals when out-of-band data is available. Positive values indicate a process ID; negative values, other than -1, indicate a process group ID. If *fil*des does not refer to a socket, the results are

fcntl()*System Interfaces*

26915

unspecified.

26916

F_SETOWN

26917

26918

26919

26920

If *fildev* refers to a socket, set the process or process group ID specified to receive SIGURG signals when out-of-band data is available, using the value of the third argument, *arg*, taken as type **int**. Positive values indicate a process ID; negative values, other than -1, indicate a process group ID. If *fildev* does not refer to a socket, the results are unspecified.

26921

The following values for *cmd* are available for advisory record locking. Record locking shall be supported for regular files, and may be supported for other files.

26922

26923

F_GETLK

26924

26925

26926

26927

26928

Get the first lock which blocks the lock description pointed to by the third argument, *arg*, taken as a pointer to type **struct flock**, defined in **<fcntl.h>**. The information retrieved shall overwrite the information passed to *fcntl()* in the structure **flock**. If no lock is found that would prevent this lock from being created, then the structure shall be left unchanged except for the lock type which shall be set to F_UNLCK.

26929

F_SETLK

26930

26931

26932

26933

26934

26935

Set or clear a file segment lock according to the lock description pointed to by the third argument, *arg*, taken as a pointer to type **struct flock**, defined in **<fcntl.h>**. F_SETLK can establish shared (or read) locks (F_RDLCK) or exclusive (or write) locks (F_WRLCK), as well as to remove either type of lock (F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are defined in **<fcntl.h>**. If a shared or exclusive lock cannot be set, *fcntl()* shall return immediately with a return value of -1.

26936

F_SETLKW

26937

26938

26939

26940

26941

This command shall be equivalent to F_SETLK except that if a shared or exclusive lock is blocked by other locks, the thread shall wait until the request can be satisfied. If a signal that is to be caught is received while *fcntl()* is waiting for a region, *fcntl()* shall be interrupted. Upon return from the signal handler, *fcntl()* shall return -1 with *errno* set to [EINTR], and the lock operation shall not be done.

26942

Additional implementation-defined values for *cmd* may be defined in **<fcntl.h>**. Their names shall start with F_.

26943

26944

When a shared lock is set on a segment of a file, other processes shall be able to set shared locks on that segment or a portion of it. A shared lock prevents any other process from setting an exclusive lock on any portion of the protected area. A request for a shared lock shall fail if the file descriptor was not opened with read access.

26945

26946

26947

26948

An exclusive lock shall prevent any other process from setting a shared lock or an exclusive lock on any portion of the protected area. A request for an exclusive lock shall fail if the file descriptor was not opened with write access.

26949

26950

26951

The structure **flock** describes the type (*l_type*), starting offset (*l_whence*), relative offset (*l_start*), size (*l_len*), and process ID (*l_pid*) of the segment of the file to be affected.

26952

26953

The value of *l_whence* is SEEK_SET, SEEK_CUR, or SEEK_END, to indicate that the relative offset *l_start* bytes shall be measured from the start of the file, current position, or end of the file, respectively. The value of *l_len* is the number of consecutive bytes to be locked. The value of *l_len* may be negative (where the definition of **off_t** permits negative values of *l_len*). The *l_pid* field is only used with F_GETLK to return the process ID of the process holding a blocking lock. After a successful F_GETLK request, when a blocking lock is found, the values returned in the **flock** structure shall be as follows:

26954

26955

26956

26957

26958

26959

System Interfaces

fcntl()

26960	<i>l_type</i>	Type of blocking lock found.
26961	<i>l_whence</i>	SEEK_SET.
26962	<i>l_start</i>	Start of the blocking lock.
26963	<i>l_len</i>	Length of the blocking lock.
26964	<i>l_pid</i>	Process ID of the process that holds the blocking lock.
26965	If the command is F_SETLK and the process must wait for another process to release a lock,	
26966	then the range of bytes to be locked shall be determined before the <i>fcntl()</i> function blocks. If the	
26967	file size or file descriptor seek offset change while <i>fcntl()</i> is blocked, this shall not affect the	
26968	range of bytes locked.	
26969	If <i>l_len</i> is positive, the area affected shall start at <i>l_start</i> and end at <i>l_start+l_len-1</i> . If <i>l_len</i> is	
26970	negative, the area affected shall start at <i>l_start+l_len</i> and end at <i>l_start-1</i> . Locks may start and	
26971	extend beyond the current end of a file, but shall not extend before the beginning of the file. A	
26972	lock shall be set to extend to the largest possible value of the file offset for that file by setting	
26973	<i>l_len</i> to 0. If such a lock also has <i>l_start</i> set to 0 and <i>l_whence</i> is set to SEEK_SET, the whole file	
26974	shall be locked.	
26975	There shall be at most one type of lock set for each byte in the file. Before a successful return	
26976	from an F_SETLK or an F_SETLKW request when the calling process has previously existing	
26977	locks on bytes in the region specified by the request, the previous lock type for each byte in the	
26978	specified region shall be replaced by the new lock type. As specified above under the	
26979	descriptions of shared locks and exclusive locks, an F_SETLK or an F_SETLKW request	
26980	(respectively) shall fail or block when another process has existing locks on bytes in the specified	
26981	region and the type of any of those locks conflicts with the type specified in the request.	
26982	All locks associated with a file for a given process shall be removed when a file descriptor for	
26983	that file is closed by that process or the process holding that file descriptor terminates. Locks are	
26984	not inherited by a child process.	
26985	A potential for deadlock occurs if a process controlling a locked region is put to sleep by	
26986	attempting to lock the locked region of another process. If the system detects that sleeping until	
26987	a locked region is unlocked would cause a deadlock, <i>fcntl()</i> shall fail with an [EDEADLK] error.	
26988	An unlock (F_UNLCK) request in which <i>l_len</i> is non-zero and the offset of the last byte of the	
26989	requested segment is the maximum value for an object of type off_t , when the process has an	
26990	existing lock in which <i>l_len</i> is 0 and which includes the last byte of the requested segment, shall	
26991	be treated as a request to unlock from the start of the requested segment with an <i>l_len</i> equal to 0.	
26992	Otherwise, an unlock (F_UNLCK) request shall attempt to unlock only the requested segment.	
26993	SHM	When the file descriptor <i>fd</i> refers to a shared memory object, the behavior of <i>fcntl()</i> shall be
26994		the same as for a regular file except the effect of the following values for the argument <i>cmd</i> shall
26995		be unspecified: F_SETFL, F_GETLK, F_SETLK, and F_SETLKW.
26996	TYM	If <i>fd</i> refers to a typed memory object, the result of the <i>fcntl()</i> function is unspecified.
26997	RETURN VALUE	
26998	Upon successful completion, the value returned shall depend on <i>cmd</i> as follows:	
26999	F_DUPFD	A new file descriptor.
27000	F_DUPFD_CLOEXEC	
27001		A new file descriptor.

fcntl()*System Interfaces*

27002	F_GETFD	Value of flags defined in <code><fcntl.h></code> . The return value shall not be negative.
27003	F_SETFD	Value other than <code>-1</code> .
27004	F_GETFL	Value of file status flags and access modes. The return value is not negative.
27005	F_SETFL	Value other than <code>-1</code> .
27006	F_GETLK	Value other than <code>-1</code> .
27007	F_SETLK	Value other than <code>-1</code> .
27008	F_SETLKW	Value other than <code>-1</code> .
27009	F_GETOWN	Value of the socket owner process or process group; this will not be <code>-1</code> .
27010	F_SETOWN	Value other than <code>-1</code> .
27011		Otherwise, <code>-1</code> shall be returned and <i>errno</i> set to indicate the error.
27012	ERRORS	
27013		The <i>fcntl()</i> function shall fail if:
27014	[EACCES] or [EAGAIN]	
27015		The <i>cmd</i> argument is <code>F_SETLK</code> ; the type of lock (<i>l_type</i>) is a shared (<code>F_RDLCK</code>) or exclusive (<code>F_WRLCK</code>) lock and the segment of a file to be locked is already exclusive-locked by another process, or the type is an exclusive lock and some portion of the segment of a file to be locked is already shared-locked or exclusive-locked by another process.
27016		
27017		
27018		
27019		
27020	[EBADF]	The <i>fildev</i> argument is not a valid open file descriptor, or the argument <i>cmd</i> is <code>F_SETLK</code> or <code>F_SETLKW</code> , the type of lock, <i>l_type</i> , is a shared lock (<code>F_RDLCK</code>), and <i>fildev</i> is not a valid file descriptor open for reading, or the type of lock, <i>l_type</i> , is an exclusive lock (<code>F_WRLCK</code>), and <i>fildev</i> is not a valid file descriptor open for writing.
27021		
27022		
27023		
27024		
27025	[EINTR]	The <i>cmd</i> argument is <code>F_SETLKW</code> and the function was interrupted by a signal.
27026	[EINVAL]	The <i>cmd</i> argument is invalid, or the <i>cmd</i> argument is <code>F_DUPFD</code> or <code>F_DUPFD_CLOEXEC</code> and <i>arg</i> is negative or greater than or equal to <code>{OPEN_MAX}</code> , or the <i>cmd</i> argument is <code>F_GETLK</code> , <code>F_SETLK</code> , or <code>F_SETLKW</code> and the data pointed to by <i>arg</i> is not valid, or <i>fildev</i> refers to a file that does not support locking.
27027		
27028		
27029		
27030		
27031	[EMFILE]	The argument <i>cmd</i> is <code>F_DUPFD</code> or <code>F_DUPFD_CLOEXEC</code> and all file descriptors available to the process are currently open, or no file descriptors greater than or equal to <i>arg</i> are available.
27032		
27033		
27034	[ENOLCK]	The argument <i>cmd</i> is <code>F_SETLK</code> or <code>F_SETLKW</code> and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.
27035		
27036		
27037	[EOVERFLOW]	One of the values to be returned cannot be represented correctly.
27038	[EOVERFLOW]	The <i>cmd</i> argument is <code>F_GETLK</code> , <code>F_SETLK</code> , or <code>F_SETLKW</code> and the smallest or, if <i>l_len</i> is non-zero, the largest offset of any byte in the requested segment cannot be represented correctly in an object of type <code>off_t</code> .
27039		
27040		

The *fcntl()* function may fail if:

[EDEADLK] The *cmd* argument is *F_SETLK*, the lock is blocked by a lock from another process, and putting the calling process to sleep to wait for that lock to become free would cause a deadlock.

EXAMPLES

Locking and Unlocking a File

The following example demonstrates how to place a lock on bytes 100 to 109 of a file and then later remove it. *F_SETLK* is used to perform a non-blocking lock request so that the process does not have to wait if an incompatible lock is held by another process; instead the process can take some other action.

```
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>

int
main(int argc, char *argv[])
{
    int fd;
    struct flock fl;

    fd = open("testfile", O_RDWR);
    if (fd == -1)
        /* Handle error */;

    /* Make a non-blocking request to place a write lock
       on bytes 100-109 of testfile */

    fl.l_type = F_WRLCK;
    fl.l_whence = SEEK_SET;
    fl.l_start = 100;
    fl.l_len = 10;

    if (fcntl(fd, F_SETLK, &fl) == -1) {
        if (errno == EACCES || errno == EAGAIN) {
            printf("Already locked by another process\n");

            /* We can't get the lock at the moment */
        } else {
            /* Handle unexpected error */;
        }
    } else { /* Lock was granted... */

        /* Perform I/O on bytes 100 to 109 of file */

        /* Unlock the locked bytes */

        fl.l_type = F_UNLCK;
        fl.l_whence = SEEK_SET;
        fl.l_start = 100;
        fl.l_len = 10;
        if (fcntl(fd, F_SETLK, &fl) == -1)
```

fcntl()

```

27085             /* Handle error */;
27086         }
27087         exit(EXIT_SUCCESS);
27088     } /* main */

```

Setting the Close-on-Exec Flag

The following example demonstrates how to set the close-on-exec flag for the file descriptor *fd*.

```

27091 #include <unistd.h>
27092 #include <fcntl.h>
27093 ...
27094     int flags;
27095
27096     flags = fcntl(fd, F_GETFD);
27097     if (flags == -1)
27098         /* Handle error */;
27099     flags |= FD_CLOEXEC;
27100     if (fcntl(fd, F_SETFD, flags) == -1)
27101         /* Handle error */;

```

APPLICATION USAGE

The *arg* values to `F_GETFD`, `F_SETFD`, `F_GETFL`, and `F_SETFL` all represent flag values to allow for future growth. Applications using these functions should do a read-modify-write operation on them, rather than assuming that only the values defined by this volume of POSIX.1-2008 are valid. It is a common error to forget this, particularly in the case of `F_SETFD`. Some implementations set additional file status flags to advise the application of default behavior, even though the application did not request these flags.

RATIONALE

The ellipsis in the SYNOPSIS is the syntax specified by the ISO C standard for a variable number of arguments. It is used because System V uses pointers for the implementation of file locking functions.

This volume of POSIX.1-2008 permits concurrent read and write access to file data using the `fcntl()` function; this is a change from the 1984 `/usr/group` standard and early proposals. Without concurrency controls, this feature may not be fully utilized without occasional loss of data.

Data losses occur in several ways. One case occurs when several processes try to update the same record, without sequencing controls; several updates may occur in parallel and the last writer “wins”. Another case is a bit-tree or other internal list-based database that is undergoing reorganization. Without exclusive use to the tree segment by the updating process, other reading processes chance getting lost in the database when the index blocks are split, condensed, inserted, or deleted. While `fcntl()` is useful for many applications, it is not intended to be overly general and does not handle the bit-tree example well.

This facility is only required for regular files because it is not appropriate for many devices such as terminals and network connections.

Since `fcntl()` works with “any file descriptor associated with that file, however it is obtained”, the file descriptor may have been inherited through a `fork()` or `exec` operation and thus may affect a file that another process also has open.

The use of the open file description to identify what to lock requires extra calls and presents problems if several processes are sharing an open file description, but there are too many

implementations of the existing mechanism for this volume of POSIX.1-2008 to use different specifications.

Another consequence of this model is that closing any file descriptor for a given file (whether or not it is the same open file description that created the lock) causes the locks on that file to be relinquished for that process. Equivalently, any close for any file/process pair relinquishes the locks owned on that file for that process. But note that while an open file description may be shared through *fork()*, locks are not inherited through *fork()*. Yet locks may be inherited through one of the *exec* functions.

The identification of a machine in a network environment is outside the scope of this volume of POSIX.1-2008. Thus, an *l_sysid* member, such as found in System V, is not included in the locking structure.

Changing of lock types can result in a previously locked region being split into smaller regions.

Mandatory locking was a major feature of the 1984 /usr/group standard.

For advisory file record locking to be effective, all processes that have access to a file must cooperate and use the advisory mechanism before doing I/O on the file. Enforcement-mode record locking is important when it cannot be assumed that all processes are cooperating. For example, if one user uses an editor to update a file at the same time that a second user executes another process that updates the same file and if only one of the two processes is using advisory locking, the processes are not cooperating. Enforcement-mode record locking would protect against accidental collisions.

Secondly, advisory record locking requires a process using locking to bracket each I/O operation with lock (or test) and unlock operations. With enforcement-mode file and record locking, a process can lock the file once and unlock when all I/O operations have been completed. Enforcement-mode record locking provides a base that can be enhanced; for example, with sharable locks. That is, the mechanism could be enhanced to allow a process to lock a file so other processes could read it, but none of them could write it.

Mandatory locks were omitted for several reasons:

1. Mandatory lock setting was done by multiplexing the set-group-ID bit in most implementations; this was confusing, at best.
2. The relationship to file truncation as supported in 4.2 BSD was not well specified.
3. Any publicly readable file could be locked by anyone. Many historical implementations keep the password database in a publicly readable file. A malicious user could thus prohibit logins. Another possibility would be to hold open a long-distance telephone line.
4. Some demand-paged historical implementations offer memory mapped files, and enforcement cannot be done on that type of file.

Since sleeping on a region is interrupted with any signal, *alarm()* may be used to provide a timeout facility in applications requiring it. This is useful in deadlock detection. Since implementation of full deadlock detection is not always feasible, the [EDEADLK] error was made optional.

FUTURE DIRECTIONS

None.

fcntl()*System Interfaces*27171 **SEE ALSO**27172 *alarm()*, *close()*, *exec*, *open()*, *sigaction()*27173 XBD **<fcntl.h>**, **<signal.h>**27174 **CHANGE HISTORY**

27175 First released in Issue 1. Derived from Issue 1 of the SVID.

27176 **Issue 5**27177 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
27178 Threads Extension.

27179 Large File Summit extensions are added.

27180 **Issue 6**27181 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.27182 The following new requirements on POSIX implementations derive from alignment with the
27183 Single UNIX Specification:

- 27184 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
27185 required for conforming implementations of previous POSIX specifications, it was not
27186 required for UNIX applications.
- 27187 • In the DESCRIPTION, sentences describing behavior when *l_len* is negative are now
27188 mandated, and the description of unlock (F_UNLOCK) when *l_len* is non-negative is
27189 mandated.
- 27190 • In the ERRORS section, the [EINVAL] error condition has the case mandated when the *cmd*
27191 is invalid, and two [EOVERFLOW] error conditions are added.

27192 The F_GETOWN and F_SETOWN values are added for sockets.

27193 The following changes were made to align with the IEEE P1003.1a draft standard:

- 27194 • Clarification is added that the extent of the bytes locked is determined prior to the
27195 blocking action.

27196 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
27197 *fcntl()* results are unspecified for typed memory objects.

27198 The normative text is updated to avoid use of the term “must” for application requirements.

27199 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/29 is applied, adding the example to the
27200 EXAMPLES section.27201 **Issue 7**27202 Austin Group Interpretation 1003.1-2001 #150 is applied, clarifying the file status flags returned
27203 when *cmd* is F_GETFL.27204 Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the
27205 FD_CLOEXEC flag atomically at *open()*, and adding the F_DUPFD_CLOEXEC flag.27206 The optional **<unistd.h>** header is removed from this function, since **<fcntl.h>** now defines
27207 SEEK_SET, SEEK_CUR, and SEEK_END as part of the Base.

27208 **NAME**27209 **fdatasync** — synchronize the data of a file (**REALTIME**)27210 **SYNOPSIS**

```
27211 SIO      #include <unistd.h>
27212          int fdatasync(int fildes);
```

27213 **DESCRIPTION**

27214 The *fdatasync()* function shall force all currently queued I/O operations associated with the file
 27215 indicated by file descriptor *fildes* to the synchronized I/O completion state.

27216 The functionality shall be equivalent to *fsync()* with the symbol `_POSIX_SYNCHRONIZED_IO`
 27217 defined, with the exception that all I/O operations shall be completed as defined for
 27218 synchronized I/O data integrity completion.

27219 **RETURN VALUE**

27220 If successful, the *fdatasync()* function shall return the value 0; otherwise, the function shall return
 27221 the value `-1` and set *errno* to indicate the error. If the *fdatasync()* function fails, outstanding I/O
 27222 operations are not guaranteed to have been completed.

27223 **ERRORS**

27224 The *fdatasync()* function shall fail if:

27225 [EBADF] The *fildes* argument is not a valid file descriptor open for writing.

27226 [EINVAL] This implementation does not support synchronized I/O for this file.

27227 In the event that any of the queued I/O operations fail, *fdatasync()* shall return the error
 27228 conditions defined for *read()* and *write()*.

27229 **EXAMPLES**

27230 None.

27231 **APPLICATION USAGE**

27232 None.

27233 **RATIONALE**

27234 None.

27235 **FUTURE DIRECTIONS**

27236 None.

27237 **SEE ALSO**

27238 *aio_fsync()*, *fcntl()*, *fsync()*, *open()*, *read()*, *write()*

27239 XBD *<unistd.h>*

27240 **CHANGE HISTORY**

27241 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

27242 **Issue 6**

27243 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 27244 implementation does not support the Synchronized Input and Output option.

27245 The *fdatasync()* function is marked as part of the Synchronized Input and Output option.

fdetach()

System Interfaces

27246 **NAME**27247 **fdetach** — detach a name from a STREAMS-based file descriptor (**STREAMS**)27248 **SYNOPSIS**

```
27249 OB XSR #include <stropts.h>
27250 int fdetach(const char *path);
```

27251 **DESCRIPTION**

27252 The *fdetach()* function shall detach a STREAMS-based file from the file to which it was attached
 27253 by a previous call to *fattach()*. The *path* argument points to the pathname of the attached
 27254 STREAMS file. The process shall have appropriate privileges or be the owner of the file. A
 27255 successful call to *fdetach()* shall cause all pathnames that named the attached STREAMS file to
 27256 again name the file to which the STREAMS file was attached. All subsequent operations on *path*
 27257 shall operate on the underlying file and not on the STREAMS file.

27258 All open file descriptions established while the STREAMS file was attached to the file referenced
 27259 by *path* shall still refer to the STREAMS file after the *fdetach()* has taken effect.

27260 If there are no open file descriptors or other references to the STREAMS file, then a successful
 27261 call to *fdetach()* shall be equivalent to performing the last *close()* on the attached file.

27262 **RETURN VALUE**

27263 Upon successful completion, *fdetach()* shall return 0; otherwise, it shall return –1 and set *errno* to
 27264 indicate the error.

27265 **ERRORS**

27266 The *fdetach()* function shall fail if:

- 27267 [EACCES] Search permission is denied on a component of the path prefix.
- 27268 [EINVAL] The *path* argument names a file that is not currently attached.
- 27269 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 27270 argument.
- 27271 [ENAMETOOLONG]
 27272 The length of a component of a pathname is longer than {NAME_MAX}.
- 27273 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 27274 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument
 27275 contains at least one non-*<slash>* character and ends with one or more trailing
 27276 *<slash>* characters and the last pathname component names an existing file
 27277 that is neither a directory nor a symbolic link to a directory.
- 27278 [EPERM] The effective user ID is not the owner of *path* and the process does not have
 27279 appropriate privileges.

27280 The *fdetach()* function may fail if:

- 27281 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 27282 resolution of the *path* argument.
- 27283 [ENAMETOOLONG]
 27284 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 27285 symbolic link produced an intermediate result with a length that exceeds
 27286 {PATH_MAX}.

27287 **EXAMPLES**27288 **Detaching a File**

27289 The following example detaches the STREAMS-based file **/tmp/named-STREAM** from the file to
 27290 which it was attached by a previous, successful call to *fattach()*. Subsequent calls to open this
 27291 file refer to the underlying file, not to the STREAMS file.

```
27292 #include <stropts.h>
27293 ...
27294     char *filename = "/tmp/named-STREAM";
27295     int ret;
27296     ret = fdetach(filename);
```

27297 **APPLICATION USAGE**

27298 None.

27299 **RATIONALE**

27300 None.

27301 **FUTURE DIRECTIONS**

27302 The *fdetach()* function may be removed in a future version.

27303 **SEE ALSO**

27304 *fattach()*

27305 XBD <stropts.h>

27306 **CHANGE HISTORY**

27307 First released in Issue 4, Version 2.

27308 **Issue 5**

27309 Moved from X/OPEN UNIX extension to BASE.

27310 **Issue 6**

27311 The normative text is updated to avoid use of the term “must” for application requirements.

27312 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
 27313 [ELOOP] error condition is added.

27314 **Issue 7**

27315 Austin Group Interpretation 1003.1-2001 #143 is applied.

27316 The *fdetach()* function is marked obsolescent.

27317 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
 27318 pathname exists but is not a directory or a symbolic link to a directory.

fdim()

System Interfaces

27319 **NAME**

27320 **fdim, fdimf, fdiml** — compute positive difference between two floating-point numbers

27321 **SYNOPSIS**

```
27322 #include <math.h>
27323 double fdim(double x, double y);
27324 float fdimf(float x, float y);
27325 long double fdiml(long double x, long double y);
```

27326 **DESCRIPTION**

27327 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 27328 conflict between the requirements described here and the ISO C standard is unintentional. This
 27329 volume of POSIX.1-2008 defers to the ISO C standard.

27330 These functions shall determine the positive difference between their arguments. If x is greater
 27331 than y , $x-y$ is returned. If x is less than or equal to y , +0 is returned.

27332 An application wishing to check for error situations should set *errno* to zero and call
 27333 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 27334 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 27335 zero, an error has occurred.

27336 **RETURN VALUE**

27337 Upon successful completion, these functions shall return the positive difference value.

27338 If $x-y$ is positive and overflows, a range error shall occur and *fdim()*, *fdimf()*, and *fdiml()* shall
 27339 return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

27340 **XSI** If $x-y$ is positive and underflows, a range error may occur, and either $(x-y)$ (if representable), or
 27341 0.0 (if supported), or an implementation-defined value shall be returned.

27342 **MX** If x or y is NaN, a NaN shall be returned.

27343 **ERRORS**

27344 The *fdim()* function shall fail if:

27345 **Range Error** The result overflows.

27346 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 27347 then *errno* shall be set to [ERANGE]. If the integer expression
 27348 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 27349 floating-point exception shall be raised.

27350 The *fdim()* function may fail if:

27351 **Range Error** The result underflows.

27352 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 27353 then *errno* shall be set to [ERANGE]. If the integer expression
 27354 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 27355 floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On implementations supporting IEEE Std 754-1985, $x-y$ cannot underflow, and hence the 0.0 return value is shaded as an extension for implementations supporting the XSI option rather than an MX extension.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *fetestexcept()*, *fmax()*, *fmin()*

Section 4.19 (on page 116), <math.h>

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

fdopen()*System Interfaces*27373 **NAME**27374 **fdopen** — associate a stream with a file descriptor27375 **SYNOPSIS**

```
27376 CX      #include <stdio.h>
27377 FILE *fdopen(int fildes, const char *mode);
```

27378 **DESCRIPTION**27379 The *fdopen()* function shall associate a stream with a file descriptor.27380 The *mode* argument is a character string having one of the following values:27381 *r* or *rb* Open a file for reading.27382 *w* or *wb* Open a file for writing.27383 *a* or *ab* Open a file for writing at end-of-file.27384 *r+* or *rb+* or *r+b* Open a file for update (reading and writing).27385 *w+* or *wb+* or *w+b* Open a file for update (reading and writing).27386 *a+* or *ab+* or *a+b* Open a file for update (reading and writing) at end-of-file.27387 The meaning of these flags is exactly as specified in *fopen()*, except that modes beginning with *w*
27388 shall not cause truncation of the file.27389 Additional values for the *mode* argument may be supported by an implementation.

27390 The application shall ensure that the mode of the stream as expressed by the *mode* argument is
27391 allowed by the file access mode of the open file description to which *fildes* refers. The file
27392 position indicator associated with the new stream is set to the position indicated by the file offset
27393 associated with the file descriptor.

27394 The error and end-of-file indicators for the stream shall be cleared. The *fdopen()* function may
27395 cause the last data access timestamp of the underlying file to be marked for update.

27396 SHM If *fildes* refers to a shared memory object, the result of the *fdopen()* function is unspecified.27397 TYM If *fildes* refers to a typed memory object, the result of the *fdopen()* function is unspecified.

27398 The *fdopen()* function shall preserve the offset maximum previously set for the open file
27399 description corresponding to *fildes*.

27400 **RETURN VALUE**

27401 Upon successful completion, *fdopen()* shall return a pointer to a stream; otherwise, a null pointer
27402 shall be returned and *errno* set to indicate the error.

27403 **ERRORS**27404 The *fdopen()* function shall fail if:

27405 [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

27406 The *fdopen()* function may fail if:27407 [EBADF] The *fildes* argument is not a valid file descriptor.27408 [EINVAL] The *mode* argument is not a valid mode.

27409 [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

27410 [ENOMEM] Insufficient space to allocate a buffer.

27411 EXAMPLES

27412 None.

27413 APPLICATION USAGE

27414 File descriptors are obtained from calls like *open()*, *dup()*, *creat()*, or *pipe()*, which open files but
27415 do not return streams.

27416 RATIONALE

27417 The file descriptor may have been obtained from *open()*, *creat()*, *pipe()*, *dup()*, *fcntl()*, or *socket()*;
27418 inherited through *fork()*, *posix_spawn()*, or *exec*; or perhaps obtained by other means.

27419 The meanings of the *mode* arguments of *fdopen()* and *fopen()* differ. With *fdopen()*, open for write
27420 (*w* or *w+*) does not truncate, and append (*a* or *a+*) cannot create for writing. The *mode* argument
27421 formats that include *a b* are allowed for consistency with the ISO C standard function *fopen()*.
27422 The *b* has no effect on the resulting stream. Although not explicitly required by this volume of
27423 POSIX.1-2008, a good implementation of append (*a*) mode would cause the O_APPEND flag to
27424 be set.

27425 FUTURE DIRECTIONS

27426 None.

27427 SEE ALSO

27428 Section 2.5.1 (on page 491), *fclose()*, *fmemopen()*, *fopen()*, *open()*, *open_memstream()*,
27429 *posix_spawn()*, *socket()*

27430 XBD <stdio.h>

27431 CHANGE HISTORY

27432 First released in Issue 1. Derived from Issue 1 of the SVID.

27433 Issue 5

27434 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

27435 Large File Summit extensions are added.

27436 Issue 6

27437 The following new requirements on POSIX implementations derive from alignment with the
27438 Single UNIX Specification:

- 27439 • In the DESCRIPTION, the use and setting of the *mode* argument are changed to include
27440 binary streams.
- 27441 • In the DESCRIPTION, text is added for large file support to indicate setting of the offset
27442 maximum in the open file description.
- 27443 • All errors identified in the ERRORS section are added.
- 27444 • In the DESCRIPTION, text is added that the *fdopen()* function may cause *st_atime* to be
27445 updated.

27446 The following changes were made to align with the IEEE P1003.1a draft standard:

- 27447 • Clarification is added that it is the responsibility of the application to ensure that the mode
27448 is compatible with the open file descriptor.

27449 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
27450 *fdopen()* results are unspecified for typed memory objects.

fdopen()*System Interfaces*

27451	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/30 is applied, making corrections to the
27452	RATIONALE.
27453	Issue 7
27454	SD5-XSH-ERN-149 is applied, adding the {STREAM_MAX} [EMFILE] error condition.
27455	Changes are made related to support for finegrained timestamps.

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

NAME

fdopendir, opendir — open directory associated with file descriptor

SYNOPSIS

```
#include <dirent.h>

DIR *fdopendir(int fd);
DIR *opendir(const char *dirname);
```

DESCRIPTION

The *fdopendir()* function shall be equivalent to the *opendir()* function except that the directory is specified by a file descriptor rather than by a name. The file offset associated with the file descriptor at the time of the call determines which entries are returned.

Upon successful return from *fdopendir()*, the file descriptor is under the control of the system, and if any attempt is made to close the file descriptor, or to modify the state of the associated description, other than by means of *closedir()*, *readdir()*, *readdir_r()*, or *rewinddir()*, the behavior is undefined. Upon calling *closedir()* the file descriptor shall be closed.

It is unspecified whether the FD_CLOEXEC flag will be set on the file descriptor by a successful call to *fdopendir()*.

The *opendir()* function shall open a directory stream corresponding to the directory named by the *dirname* argument. The directory stream is positioned at the first entry. If the type **DIR** is implemented using a file descriptor, applications shall only be able to open up to a total of {OPEN_MAX} files and directories.

If the type **DIR** is implemented using a file descriptor, the descriptor shall be obtained as if the O_DIRECTORY flag was passed to *open()*.

RETURN VALUE

Upon successful completion, these functions shall return a pointer to an object of type **DIR**. Otherwise, these functions shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *fdopendir()* function shall fail if:

[EBADF] The *fd* argument is not a valid file descriptor open for reading.

[ENOTDIR] The descriptor *fd* is not associated with a directory.

The *opendir()* function shall fail if:

[EACCES] Search permission is denied for the component of the path prefix of *dirname* or read permission is denied for *dirname*.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *dirname* argument.

[ENAMETOOLONG]

The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *dirname* does not name an existing directory or *dirname* is an empty string.

[ENOTDIR] A component of *dirname* is not a directory.

fdopendir()*System Interfaces*

- 27495 The *opendir()* function may fail if:
- 27496 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
27497 resolution of the *dirname* argument.
- 27498 [EMFILE] All file descriptors available to the process are currently open.
- 27499 [ENAMETOOLONG]
27500 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
27501 symbolic link produced an intermediate result with a length that exceeds
27502 {PATH_MAX}.
- 27503 [ENFILE] Too many files are currently open in the system.

EXAMPLES**Open a Directory Stream**

27506 The following program fragment demonstrates how the *opendir()* function is used.

```
27507 #include <dirent.h>
27508 ...
27509     DIR *dir;
27510     struct dirent *dp;
27511 ...
27512     if ((dir = opendir (".")) == NULL) {
27513         perror ("Cannot open .");
27514         exit (1);
27515     }
27516     while ((dp = readdir (dir)) != NULL) {
27517 ...
```

Find And Open a File

27519 The following program searches through a given directory looking for files whose name does
27520 not begin with a dot and whose size is larger than 1 MiB.

```
27521 #include <stdio.h>
27522 #include <dirent.h>
27523 #include <fcntl.h>
27524 #include <sys/stat.h>
27525 #include <stdint.h>
27526 #include <stdlib.h>
27527 #include <unistd.h>
27528 int
27529 main(int argc, char *argv[])
27530 {
27531     struct stat statbuf;
27532     DIR *d;
27533     struct dirent *dp;
27534     int dfd, ffd;
27535     if ((d = fdopendir((dfd = open("./tmp", O_RDONLY))) == NULL) {
27536         fprintf(stderr, "Cannot open ./tmp directory\n");
27537         exit(1);
```

```

27538     }
27539     while ((dp = readdir(d)) != NULL) {
27540         if (dp->d_name[0] == '.')
27541             continue;
27542         /* there is a possible race condition here as the file
27543          * could be renamed between the readdir and the open */
27544         if ((ffd = openat(dfd, dp->d_name, O_RDONLY)) == -1) {
27545             perror(dp->d_name);
27546             continue;
27547         }
27548         if (fstat(ffd, &statbuf) == 0 && statbuf.st_size > (1024*1024)) {
27549             /* found it ... */
27550             printf("%s: %jdK\n", dp->d_name,
27551                 (intmax_t)(statbuf.st_size / 1024));
27552         }
27553         close(ffd);
27554     }
27555     closedir(d); // note this implicitly closes dfd
27556     return 0;
27557 }

```

APPLICATION USAGE

The *opendir()* function should be used in conjunction with *readdir()*, *closedir()*, and *rewinddir()* to examine the contents of the directory (see the EXAMPLES section in *readdir()*). This method is recommended for portability.

RATIONALE

The purpose of the *fdopendir()* function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *opendir()*, resulting in unspecified behavior.

Based on historical implementations, the rules about file descriptors apply to directory streams as well. However, this volume of POSIX.1-2008 does not mandate that the directory stream be implemented using file descriptors. The description of *closedir()* clarifies that if a file descriptor is used for the directory stream, it is mandatory that *closedir()* deallocate the file descriptor. When a file descriptor is used to implement the directory stream, it behaves as if the *FD_CLOEXEC* had been set for the file descriptor.

The directory entries for dot and dot-dot are optional. This volume of POSIX.1-2008 does not provide a way to test *a priori* for their existence because an application that is portable must be written to look for (and usually ignore) those entries. Writing code that presumes that they are the first two entries does not always work, as many implementations permit them to be other than the first two entries, with a “normal” entry preceding them. There is negligible value in providing a way to determine what the implementation does because the code to deal with dot and dot-dot must be written in any case and because such a flag would add to the list of those flags (which has proven in itself to be objectionable) and might be abused.

Since the structure and buffer allocation, if any, for directory operations are defined by the implementation, this volume of POSIX.1-2008 imposes no portability requirements for erroneous program constructs, erroneous data, or the use of unspecified values such as the use or referencing of a *dirp* value or a **dirent** structure value after a directory stream has been closed or after a *fork()* or one of the *exec* function calls.

fdopendir()

System Interfaces

27585 **FUTURE DIRECTIONS**

27586 None.

27587 **SEE ALSO**27588 *closedir()*, *dirfd()*, *fstatat()*, *open()*, *readdir()*, *rewinddir()*, *symlink()*27589 XBD **<dirent.h>**, **<sys/types.h>**27590 **CHANGE HISTORY**

27591 First released in Issue 2.

27592 **Issue 6**27593 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.27594 The following new requirements on POSIX implementations derive from alignment with the
27595 Single UNIX Specification:

- 27596 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
27597 required for conforming implementations of previous POSIX specifications, it was not
27598 required for UNIX applications.
- 27599 • The [ELOOP] mandatory error condition is added.
- 27600 • A second [ENAMETOOLONG] is added as an optional error condition.

27601 The following changes were made to align with the IEEE P1003.1a draft standard:

- 27602 • The [ELOOP] optional error condition is added.

27603 **Issue 7**

27604 Austin Group Interpretation 1003.1-2001 #143 is applied.

27605 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

27606 The *fdopendir()* function is added from The Open Group Technical Standard, 2006, Extended API
27607 Set Part 2.

27608 An additional example is added.

27609 **NAME**27610 `feclearexcept` — clear floating-point exception27611 **SYNOPSIS**27612 `#include <fenv.h>`27613 `int feclearexcept(int excepts);`27614 **DESCRIPTION**

27615 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 27616 conflict between the requirements described here and the ISO C standard is unintentional. This
 27617 volume of POSIX.1-2008 defers to the ISO C standard.

27618 The `feclearexcept()` function shall attempt to clear the supported floating-point exceptions
 27619 represented by `excepts`.

27620 **RETURN VALUE**

27621 If the argument is zero or if all the specified exceptions were successfully cleared, `feclearexcept()`
 27622 shall return zero. Otherwise, it shall return a non-zero value.

27623 **ERRORS**

27624 No errors are defined.

27625 **EXAMPLES**

27626 None.

27627 **APPLICATION USAGE**

27628 None.

27629 **RATIONALE**

27630 None.

27631 **FUTURE DIRECTIONS**

27632 None.

27633 **SEE ALSO**27634 [*fegetexceptflag\(\)*](#), [*feraiseexcept\(\)*](#), [*fetestexcept\(\)*](#)27635 XBD [*<fenv.h>*](#)27636 **CHANGE HISTORY**

27637 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27638 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

fegetenv()27639 **NAME**27640 `fegetenv`, `fesetenv` — get and set current floating-point environment27641 **SYNOPSIS**27642 `#include <fenv.h>`27643 `int fegetenv(fenv_t *envp);`27644 `int fesetenv(const fenv_t *envp);`27645 **DESCRIPTION**

27646 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 27647 conflict between the requirements described here and the ISO C standard is unintentional. This
 27648 volume of POSIX.1-2008 defers to the ISO C standard.

27649 The *fegetenv()* function shall attempt to store the current floating-point environment in the object
 27650 pointed to by *envp*.

27651 The *fesetenv()* function shall attempt to establish the floating-point environment represented by
 27652 the object pointed to by *envp*. The argument *envp* shall point to an object set by a call to
 27653 *fegetenv()* or *feholdexcept()*, or equal a floating-point environment macro. The *fesetenv()* function
 27654 does not raise floating-point exceptions, but only installs the state of the floating-point status
 27655 flags represented through its argument.

27656 **RETURN VALUE**

27657 If the representation was successfully stored, *fegetenv()* shall return zero. Otherwise, it shall
 27658 return a non-zero value. If the environment was successfully established, *fesetenv()* shall return
 27659 zero. Otherwise, it shall return a non-zero value.

27660 **ERRORS**

27661 No errors are defined.

27662 **EXAMPLES**

27663 None.

27664 **APPLICATION USAGE**

27665 None.

27666 **RATIONALE**

27667 None.

27668 **FUTURE DIRECTIONS**

27669 None.

27670 **SEE ALSO**27671 *feholdexcept()*, *feupdateenv()*

27672 XBD <fenv.h>

27673 **CHANGE HISTORY**

27674 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27675 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

NAME

fegetexceptflag, fesetexceptflag — get and set floating-point status flags

SYNOPSIS

```
#include <fenv.h>
```

```
int fegetexceptflag(fexcept_t *flagp, int excepts);
int fesetexceptflag(const fexcept_t *flagp, int excepts);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *fegetexceptflag()* function shall attempt to store an implementation-defined representation of the states of the floating-point status flags indicated by the argument *excepts* in the object pointed to by the argument *flagp*.

The *fesetexceptflag()* function shall attempt to set the floating-point status flags indicated by the argument *excepts* to the states stored in the object pointed to by *flagp*. The value pointed to by *flagp* shall have been set by a previous call to *fegetexceptflag()* whose second argument represented at least those floating-point exceptions represented by the argument *excepts*. This function does not raise floating-point exceptions, but only sets the state of the flags.

RETURN VALUE

If the representation was successfully stored, *fegetexceptflag()* shall return zero. Otherwise, it shall return a non-zero value. If the *excepts* argument is zero or if all the specified exceptions were successfully set, *fesetexceptflag()* shall return zero. Otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *feraiseexcept()*, *fetestexcept()*

XBD *<fenv.h>*

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

fegetround()

System Interfaces

27715 **NAME**

27716 fegetround, fesetround — get and set current rounding direction

27717 **SYNOPSIS**

```
27718 #include <fenv.h>
27719 int fegetround(void);
27720 int fesetround(int round);
```

27721 **DESCRIPTION**

27722 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27723 conflict between the requirements described here and the ISO C standard is unintentional. This
 27724 volume of POSIX.1-2008 defers to the ISO C standard.

27725 The *fegetround()* function shall get the current rounding direction.

27726 The *fesetround()* function shall establish the rounding direction represented by its argument
 27727 *round*. If the argument is not equal to the value of a rounding direction macro, the rounding
 27728 direction is not changed.

27729 **RETURN VALUE**

27730 The *fegetround()* function shall return the value of the rounding direction macro representing the
 27731 current rounding direction or a negative value if there is no such rounding direction macro or
 27732 the current rounding direction is not determinable.

27733 The *fesetround()* function shall return a zero value if and only if the requested rounding direction
 27734 was established.

27735 **ERRORS**

27736 No errors are defined.

27737 **EXAMPLES**

27738 The following example saves, sets, and restores the rounding direction, reporting an error and
 27739 aborting if setting the rounding direction fails:

```
27740 #include <fenv.h>
27741 #include <assert.h>
27742 void f(int round_dir)
27743 {
27744     #pragma STDC FENV_ACCESS ON
27745     int save_round;
27746     int setround_ok;
27747     save_round = fegetround();
27748     setround_ok = fesetround(round_dir);
27749     assert(setround_ok == 0);
27750     /* ... */
27751     fesetround(save_round);
27752     /* ... */
27753 }
```

27754 **APPLICATION USAGE**

27755 None.

27756 **RATIONALE**

27757 None.

27758 **FUTURE DIRECTIONS**

27759 None.

27760 **SEE ALSO**27761 XBD [<fenv.h>](#)27762 **CHANGE HISTORY**

27763 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

27764 ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

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

feholdexcept()

System Interfaces

27765 **NAME**

27766 feholdexcept — save current floating-point environment

27767 **SYNOPSIS**

27768 #include <fenv.h>

27769 int feholdexcept(fenv_t *envp);

27770 **DESCRIPTION**

27771 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27772 conflict between the requirements described here and the ISO C standard is unintentional. This
 27773 volume of POSIX.1-2008 defers to the ISO C standard.

27774 The *feholdexcept()* function shall save the current floating-point environment in the object
 27775 pointed to by *envp*, clear the floating-point status flags, and then install a non-stop (continue on
 27776 floating-point exceptions) mode, if available, for all floating-point exceptions.

27777 **RETURN VALUE**

27778 The *feholdexcept()* function shall return zero if and only if non-stop floating-point exception
 27779 handling was successfully installed.

27780 **ERRORS**

27781 No errors are defined.

27782 **EXAMPLES**

27783 None.

27784 **APPLICATION USAGE**

27785 None.

27786 **RATIONALE**

27787 The *feholdexcept()* function should be effective on typical IEC 60559:1989 standard
 27788 implementations which have the default non-stop mode and at least one other mode for trap
 27789 handling or aborting. If the implementation provides only the non-stop mode, then installing the
 27790 non-stop mode is trivial.

27791 **FUTURE DIRECTIONS**

27792 None.

27793 **SEE ALSO**27794 *fegetenv()*, *feupdateenv()*

27795 XBD <fenv.h>

27796 **CHANGE HISTORY**

27797 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27798 **NAME**

27799 feof — test end-of-file indicator on a stream

27800 **SYNOPSIS**

27801 #include <stdio.h>

27802 int feof(FILE *stream);

27803 **DESCRIPTION**

27804 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27805 conflict between the requirements described here and the ISO C standard is unintentional. This
 27806 volume of POSIX.1-2008 defers to the ISO C standard.

27807 The *feof()* function shall test the end-of-file indicator for the stream pointed to by *stream*.27808 **RETURN VALUE**27809 The *feof()* function shall return non-zero if and only if the end-of-file indicator is set for *stream*.27810 **ERRORS**

27811 No errors are defined.

27812 **EXAMPLES**

27813 None.

27814 **APPLICATION USAGE**

27815 None.

27816 **RATIONALE**

27817 None.

27818 **FUTURE DIRECTIONS**

27819 None.

27820 **SEE ALSO**27821 *clearerr()*, *ferror()*, *fopen()*

27822 XBD <stdio.h>

27823 **CHANGE HISTORY**

27824 First released in Issue 1. Derived from Issue 1 of the SVID.

feraiseexcept()

System Interfaces

27825 **NAME**27826 **feraiseexcept** — raise floating-point exception27827 **SYNOPSIS**27828 `#include <fenv.h>`27829 `int feraiseexcept(int excepts);`27830 **DESCRIPTION**

27831 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 27832 conflict between the requirements described here and the ISO C standard is unintentional. This
 27833 volume of POSIX.1-2008 defers to the ISO C standard.

27834 The *feraiseexcept()* function shall attempt to raise the supported floating-point exceptions
 27835 represented by the argument *excepts*. The order in which these floating-point exceptions are
 27836 raised is unspecified. Whether the *feraiseexcept()* function additionally raises the inexact floating-
 27837 point exception whenever it raises the overflow or underflow floating-point exception is
 27838 implementation-defined.

27839 **RETURN VALUE**

27840 If the argument is zero or if all the specified exceptions were successfully raised, *feraiseexcept()*
 27841 shall return zero. Otherwise, it shall return a non-zero value.

27842 **ERRORS**

27843 No errors are defined.

27844 **EXAMPLES**

27845 None.

27846 **APPLICATION USAGE**

27847 The effect is intended to be similar to that of floating-point exceptions raised by arithmetic
 27848 operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.

27849 **RATIONALE**

27850 Raising overflow or underflow is allowed to also raise inexact because on some architectures the
 27851 only practical way to raise an exception is to execute an instruction that has the exception as a
 27852 side-effect. The function is not restricted to accept only valid coincident expressions for atomic
 27853 operations, so the function can be used to raise exceptions accrued over several operations.

27854 **FUTURE DIRECTIONS**

27855 None.

27856 **SEE ALSO**27857 *feclearexcept()*, *fegetexceptflag()*, *fetestexcept()*27858 XBD `<fenv.h>`27859 **CHANGE HISTORY**

27860 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27861 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

27862 **NAME**27863 **ferror** — test error indicator on a stream27864 **SYNOPSIS**27865 `#include <stdio.h>`27866 `int ferror(FILE *stream);`27867 **DESCRIPTION**

27868 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27869 conflict between the requirements described here and the ISO C standard is unintentional. This
 27870 volume of POSIX.1-2008 defers to the ISO C standard.

27871 The *ferror()* function shall test the error indicator for the stream pointed to by *stream*.27872 **RETURN VALUE**27873 The *ferror()* function shall return non-zero if and only if the error indicator is set for *stream*.27874 **ERRORS**

27875 No errors are defined.

27876 **EXAMPLES**

27877 None.

27878 **APPLICATION USAGE**

27879 None.

27880 **RATIONALE**

27881 None.

27882 **FUTURE DIRECTIONS**

27883 None.

27884 **SEE ALSO**27885 *clearerr()*, *feof()*, *fopen()*27886 XBD `<stdio.h>`27887 **CHANGE HISTORY**

27888 First released in Issue 1. Derived from Issue 1 of the SVID.

fesetenv()*System Interfaces*27889 **NAME**

27890 fesetenv — set current floating-point environment

27891 **SYNOPSIS**

27892 #include <fenv.h>

27893 int fesetenv(const fenv_t *envp);

27894 **DESCRIPTION**27895 Refer to *fegetenv()*.

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

*System Interfaces***fesetexceptflag()**27896 **NAME**

27897 fesetexceptflag — set floating-point status flags

27898 **SYNOPSIS**

27899 #include <fenv.h>

27900 int fesetexceptflag(const fexcept_t *flagp, int excepts);

27901 **DESCRIPTION**27902 Refer to *fegetexceptflag()*.

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

fesetround()*System Interfaces*27903 **NAME**

27904 fesetround — set current rounding direction

27905 **SYNOPSIS**

27906 #include <fenv.h>

27907 int fesetround(int *round*);27908 **DESCRIPTION**27909 Refer to *fegetround()*.

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

NAME

fetestexcept — test floating-point exception flags

SYNOPSIS

#include <fenv.h>

int fetestexcept(int *excepts*);**DESCRIPTION**

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *fetestexcept()* function shall determine which of a specified subset of the floating-point exception flags are currently set. The *excepts* argument specifies the floating-point status flags to be queried.

RETURN VALUE

The *fetestexcept()* function shall return the value of the bitwise-inclusive OR of the floating-point exception macros corresponding to the currently set floating-point exceptions included in *excepts*.

ERRORS

No errors are defined.

EXAMPLES

The following example calls function *f()* if an invalid exception is set, and then function *g()* if an overflow exception is set:

```
#include <fenv.h>
/* ... */
{
    #pragma STDC FENV_ACCESS ON
    int set_excepts;
    feclearexcept (FE_INVALID | FE_OVERFLOW);
    // maybe raise exceptions
    set_excepts = fetestexcept (FE_INVALID | FE_OVERFLOW);
    if (set_excepts & FE_INVALID) f();
    if (set_excepts & FE_OVERFLOW) g();
    /* ... */
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO*feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*

XBD <fenv.h>

fetestexcept()

System Interfaces

CHANGE HISTORY

27952
27953

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

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

NAME

feupdateenv — update floating-point environment

SYNOPSIS

```
#include <fenv.h>

int feupdateenv(const fenv_t *envp);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *feupdateenv()* function shall attempt to save the currently raised floating-point exceptions in its automatic storage, attempt to install the floating-point environment represented by the object pointed to by *envp*, and then attempt to raise the saved floating-point exceptions. The argument *envp* shall point to an object set by a call to *feholdexcept()* or *fegetenv()*, or equal a floating-point environment macro.

RETURN VALUE

The *feupdateenv()* function shall return a zero value if and only if all the required actions were successfully carried out.

ERRORS

No errors are defined.

EXAMPLES

The following example shows sample code to hide spurious underflow floating-point exceptions:

```
#include <fenv.h>
double f(double x)
{
    #pragma STDC FENV_ACCESS ON
    double result;
    fenv_t save_env;
    feholdexcept(&save_env);
    // compute result
    if (/* test spurious underflow */)
        feclearexcept(FE_UNDERFLOW);
    feupdateenv(&save_env);
    return result;
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fegetenv(), *feholdexcept()*

XBD *<fenv.h>*

feupdateenv()*System Interfaces***CHANGE HISTORY**

27998	
27999	First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.
28000	ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

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

*System Interfaces***fexecve**28001 **NAME**

28002 fexecve — execute a file

28003 **SYNOPSIS**

28004 #include <unistd.h>

28005 int fexecve(int *fd*, char *const *argv*[], char *const *envp*[]);28006 **DESCRIPTION**28007 Refer to *exec*.

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

fflush()*System Interfaces*28008 **NAME**

28009 fflush — flush a stream

28010 **SYNOPSIS**

28011 #include <stdio.h>

28012 int fflush(FILE *stream);

28013 **DESCRIPTION**

28014 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28015 conflict between the requirements described here and the ISO C standard is unintentional. This
 28016 volume of POSIX.1-2008 defers to the ISO C standard.

28017 If *stream* points to an output stream or an update stream in which the most recent operation was
 28018 CX not input, *fflush()* shall cause any unwritten data for that stream to be written to the file, and the
 28019 last data modification and last file status change timestamps of the underlying file shall be
 28020 marked for update.

28021 If *stream* is a null pointer, *fflush()* shall perform this flushing action on all streams for which the
 28022 behavior is defined above.

28023 CX For a stream open for reading, if the file is not already at EOF, and the file is one capable of
 28024 seeking, the file offset of the underlying open file description shall be adjusted so that the next
 28025 operation on the open file description deals with the byte after the last one read from or written
 28026 to the stream being flushed.

28027 **RETURN VALUE**

28028 Upon successful completion, *fflush()* shall return 0; otherwise, it shall set the error indicator for
 28029 CX the stream, return EOF, and set *errno* to indicate the error.

28030 **ERRORS**28031 The *fflush()* function shall fail if:

28032 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and
 28033 the thread would be delayed in the write operation.

28034 CX [EBADF] The file descriptor underlying *stream* is not valid.

28035 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

28036 XSI [EFBIG] An attempt was made to write a file that exceeds the file size limit of the
 28037 process.

28038 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 28039 offset maximum associated with the corresponding stream.

28040 CX [EINTR] The *fflush()* function was interrupted by a signal.

28041 CX [EIO] The process is a member of a background process group attempting to write to
 28042 its controlling terminal, TOSTOP is set, the process is neither ignoring nor
 28043 blocking SIGTTOU, and the process group of the process is orphaned. This
 28044 error may also be returned under implementation-defined conditions.

28045 CX [ENOMEM] The underlying stream was created by *open_memstream()* or
 28046 *open_wmemstream()* and insufficient memory is available.

28047 CX [ENOSPC] There was no free space remaining on the device containing the file or in the
 28048 buffer used by the *fmemopen()* function.

28049 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 28050 any process. A SIGPIPE signal shall also be sent to the thread.

28051 The *fflush()* function may fail if:

28052 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 28053 capabilities of the device.

28054 EXAMPLES

28055 Sending Prompts to Standard Output

28056 The following example uses *printf()* calls to print a series of prompts for information the user
 28057 must enter from standard input. The *fflush()* calls force the output to standard output. The
 28058 *fflush()* function is used because standard output is usually buffered and the prompt may not
 28059 immediately be printed on the output or terminal. The *getline()* function calls read strings from
 28060 standard input and place the results in variables, for use later in the program.

```

28061 char *user;
28062 char *oldpasswd;
28063 char *newpasswd;
28064 ssize_t llen;
28065 size_t blen;
28066 struct termios term;
28067 tcflag_t saveflag;

28068 printf("User name: ");
28069 fflush(stdout);
28070 blen = 0;
28071 llen = getline(&user, &blen, stdin);
28072 user[llen-1] = 0;
28073 tcgetattr(fileno(stdin), &term);
28074 saveflag = term.c_lflag;
28075 term.c_lflag &= ~ECHO;
28076 tcsetattr(fileno(stdin), TCSANOW, &term);
28077 printf("Old password: ");
28078 fflush(stdout);
28079 blen = 0;
28080 llen = getline(&oldpasswd, &blen, stdin);
28081 oldpasswd[llen-1] = 0;

28082 printf("\nNew password: ");
28083 fflush(stdout);
28084 blen = 0;
28085 llen = getline(&newpasswd, &blen, stdin);
28086 newpasswd[llen-1] = 0;
28087 term.c_lflag = saveflag;
28088 tcsetattr(fileno(stdin), TCSANOW, &term);
28089 free(user);
28090 free(oldpasswd);
28091 free(newpasswd);

```

fflush()*System Interfaces*28092 **APPLICATION USAGE**

28093 None.

28094 **RATIONALE**

28095 Data buffered by the system may make determining the validity of the position of the current
 28096 file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after *fflush()*
 28097 on streams open for *read()* is not mandated by POSIX.1-2008.

28098 **FUTURE DIRECTIONS**

28099 None.

28100 **SEE ALSO**28101 *fmemopen(), getrlimit(), open_memstream(), ulimit()*28102 XBD *<stdio.h>*28103 **CHANGE HISTORY**

28104 First released in Issue 1. Derived from Issue 1 of the SVID.

28105 **Issue 5**

28106 Large File Summit extensions are added.

28107 **Issue 6**

28108 Extensions beyond the ISO C standard are marked.

28109 The following new requirements on POSIX implementations derive from alignment with the
 28110 Single UNIX Specification:

- 28111 • The [EFBIG] error is added as part of the large file support extensions.
- 28112 • The [ENXIO] optional error condition is added.

28113 The RETURN VALUE section is updated to note that the error indicator shall be set for the
 28114 stream. This is for alignment with the ISO/IEC 9899:1999 standard.

28115 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/31 is applied, updating the [EAGAIN]
 28116 error in the ERRORS section from “the process would be delayed” to “the thread would be
 28117 delayed”.

28118 **Issue 7**

28119 Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file
 28120 descriptors and streams.

28121 The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open
 28122 Group Technical Standard, 2006, Extended API Set Part 1.

28123 The EXAMPLES section is revised.

28124 Changes are made related to support for finegrained timestamps.

*System Interfaces***ffs()**28125 **NAME**

28126 ffs — find first set bit

28127 **SYNOPSIS**

28128 XSI #include <strings.h>

28129 int ffs(int i);

28130 **DESCRIPTION**

28131 The *ffs()* function shall find the first bit set (beginning with the least significant bit) in *i*, and
 28132 return the index of that bit. Bits are numbered starting at one (the least significant bit).

28133 **RETURN VALUE**28134 The *ffs()* function shall return the index of the first bit set. If *i* is 0, then *ffs()* shall return 0.28135 **ERRORS**

28136 No errors are defined.

28137 **EXAMPLES**

28138 None.

28139 **APPLICATION USAGE**

28140 None.

28141 **RATIONALE**

28142 None.

28143 **FUTURE DIRECTIONS**

28144 None.

28145 **SEE ALSO**

28146 XBD <strings.h>

28147 **CHANGE HISTORY**

28148 First released in Issue 4, Version 2.

28149 **Issue 5**

28150 Moved from X/OPEN UNIX extension to BASE.

fgetc()*System Interfaces*28151 **NAME**28152 `fgetc` — get a byte from a stream28153 **SYNOPSIS**28154 `#include <stdio.h>`28155 `int fgetc(FILE *stream);`28156 **DESCRIPTION**

28157 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28158 conflict between the requirements described here and the ISO C standard is unintentional. This
 28159 volume of POSIX.1-2008 defers to the ISO C standard.

28160 If the end-of-file indicator for the input stream pointed to by *stream* is not set and a next byte is
 28161 present, the *fgetc()* function shall obtain the next byte as an **unsigned char** converted to an **int**,
 28162 from the input stream pointed to by *stream*, and advance the associated file position indicator for
 28163 the stream (if defined). Since *fgetc()* operates on bytes, reading a character consisting of multiple
 28164 bytes (or “a multi-byte character”) may require multiple calls to *fgetc()*.

28165 CX The *fgetc()* function may mark the last data access timestamp of the file associated with *stream*
 28166 for update. The last data access timestamp shall be marked for update by the first successful
 28167 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*, or
 28168 *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

28169 **RETURN VALUE**

28170 Upon successful completion, *fgetc()* shall return the next byte from the input stream pointed to
 28171 by *stream*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the
 28172 end-of-file indicator for the stream shall be set and *fgetc()* shall return EOF. If a read error occurs,
 28173 CX the error indicator for the stream shall be set, *fgetc()* shall return EOF, and shall set *errno* to
 28174 indicate the error.

28175 **ERRORS**28176 The *fgetc()* function shall fail if data needs to be read and:

28177 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and
 28178 the thread would be delayed in the *fgetc()* operation.

28179 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
 28180 reading.

28181 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data
 28182 was transferred.

28183 CX [EIO] A physical I/O error has occurred, or the process is in a background process
 28184 group attempting to read from its controlling terminal, and either the process
 28185 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.
 28186 This error may also be generated for implementation-defined reasons.

28187 CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the
 28188 offset maximum associated with the corresponding stream.

28189 The *fgetc()* function may fail if:

28190 CX [ENOMEM] Insufficient storage space is available.

28191 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 28192 capabilities of the device.

EXAMPLES

None.

APPLICATION USAGE

If the integer value returned by *fgetc()* is stored into a variable of type **char** and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type **char** on widening to integer is implementation-defined.

The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an end-of-file condition.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feof(), *ferror()*, *fgets()*, *fread()*, *fscanf()*, *getchar()*, *getc()*, *gets()*, *ungetc()*

XBD [<stdio.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The DESCRIPTION is updated to clarify the behavior when the end-of-file indicator for the input stream is not set.
- The RETURN VALUE section is updated to note that the error indicator shall be set for the stream.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/32 is applied, updating the [EAGAIN] error in the ERRORS section from “the process would be delayed” to “the thread would be delayed”.

Issue 7

Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark the last data access timestamp for update.

fgetpos()*System Interfaces*28229 **NAME**28230 **fgetpos** — get current file position information28231 **SYNOPSIS**28232 `#include <stdio.h>`28233 `int fgetpos(FILE *restrict stream, fpos_t *restrict pos);`28234 **DESCRIPTION**

28235 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28236 conflict between the requirements described here and the ISO C standard is unintentional. This
 28237 volume of POSIX.1-2008 defers to the ISO C standard.

28238 The *fgetpos()* function shall store the current values of the parse state (if any) and file position
 28239 indicator for the stream pointed to by *stream* in the object pointed to by *pos*. The value stored
 28240 contains unspecified information usable by *fsetpos()* for repositioning the stream to its position
 28241 at the time of the call to *fgetpos()*.

28242 **RETURN VALUE**

28243 Upon successful completion, *fgetpos()* shall return 0; otherwise, it shall return a non-zero value
 28244 and set *errno* to indicate the error.

28245 **ERRORS**28246 The *fgetpos()* function shall fail if:

28247 CX [EOVERFLOW] The current value of the file position cannot be represented correctly in an
 28248 object of type **fpos_t**.

28249 The *fgetpos()* function may fail if:

28250 CX [EBADF] The file descriptor underlying *stream* is not valid.

28251 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

28252 **EXAMPLES**

28253 None.

28254 **APPLICATION USAGE**

28255 None.

28256 **RATIONALE**

28257 None.

28258 **FUTURE DIRECTIONS**

28259 None.

28260 **SEE ALSO**28261 *fopen()*, *ftell()*, *rewind()*, *ungetc()*28262 XBD `<stdio.h>`28263 **CHANGE HISTORY**

28264 First released in Issue 4. Derived from the ISO C standard.

28265 **Issue 5**

28266 Large File Summit extensions are added.

28267 **Issue 6**

28268 Extensions beyond the ISO C standard are marked.

28269 The following new requirements on POSIX implementations derive from alignment with the
28270 Single UNIX Specification:

- 28271 • The [EBADF] and [ESPIPE] optional error conditions are added.

28272 An additional [ESPIPE] error condition is added for sockets.

28273 The prototype for *fgetpos()* is changed for alignment with the ISO/IEC 9899:1999 standard.

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

fgets()28274 **NAME**28275 `fgets` — get a string from a stream28276 **SYNOPSIS**28277 `#include <stdio.h>`28278 `char *fgets(char *restrict s, int n, FILE *restrict stream);`28279 **DESCRIPTION**

28280 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28281 conflict between the requirements described here and the ISO C standard is unintentional. This
 28282 volume of POSIX.1-2008 defers to the ISO C standard.

28283 The `fgets()` function shall read bytes from *stream* into the array pointed to by *s*, until *n*–1 bytes
 28284 are read, or a <newline> is read and transferred to *s*, or an end-of-file condition is encountered.
 28285 The string is then terminated with a null byte.

28286 CX The `fgets()` function may mark the last data access timestamp of the file associated with *stream*
 28287 for update. The last data access timestamp shall be marked for update by the first successful
 28288 execution of `fgetc()`, `fgets()`, `fread()`, `fscanf()`, `getc()`, `getchar()`, `getdelim()`, `getline()`, `gets()`, or
 28289 `scanf()` using *stream* that returns data not supplied by a prior call to `ungetc()`.

28290 **RETURN VALUE**

28291 Upon successful completion, `fgets()` shall return *s*. If the stream is at end-of-file, the end-of-file
 28292 indicator for the stream shall be set and `fgets()` shall return a null pointer. If a read error occurs,
 28293 CX the error indicator for the stream shall be set, `fgets()` shall return a null pointer, and shall set
 28294 `errno` to indicate the error.

28295 **ERRORS**28296 Refer to `fgetc()`.28297 **EXAMPLES**28298 **Reading Input**

28299 The following example uses `fgets()` to read each line of input. {LINE_MAX}, which defines the
 28300 maximum size of the input line, is defined in the <limits.h> header.

```

28301 #include <stdio.h>
28302 ...
28303 char line[LINE_MAX];
28304 ...
28305 while (fgets(line, LINE_MAX, fp) != NULL) {
28306     ...
28307 }
28308 ...

```

28309 **APPLICATION USAGE**

28310 None.

28311 **RATIONALE**

28312 None.

28313 **FUTURE DIRECTIONS**

28314 None.

28315 **SEE ALSO**28316 *fprintf()*, *fopen()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *gets()*, *ungetc()*28317 XBD **<stdio.h>**28318 **CHANGE HISTORY**

28319 First released in Issue 1. Derived from Issue 1 of the SVID.

28320 **Issue 6**

28321 Extensions beyond the ISO C standard are marked.

28322 The prototype for *fprintf()* is changed for alignment with the ISO/IEC 9899:1999 standard.28323 **Issue 7**28324 Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark
28325 the last data access timestamp for update.

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

fgetwc()*System Interfaces*28326 **NAME**28327 **fgetwc** — get a wide-character code from a stream28328 **SYNOPSIS**28329 `#include <stdio.h>`28330 `#include <wchar.h>`28331 `wint_t fgetwc(FILE *stream);`28332 **DESCRIPTION**

28333 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28334 conflict between the requirements described here and the ISO C standard is unintentional. This
 28335 volume of POSIX.1-2008 defers to the ISO C standard.

28336 The *fgetwc()* function shall obtain the next character (if present) from the input stream pointed to
 28337 by *stream*, convert that to the corresponding wide-character code, and advance the associated file
 28338 position indicator for the stream (if defined).

28339 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

28340 CX The *fgetwc()* function may mark the last data access timestamp of the file associated with *stream*
 28341 for update. The last data access timestamp shall be marked for update by the first successful
 28342 execution of *fgetwc()*, *fgetws()*, *fwscanf()*, *getwc()*, *getwchar()*, *ofwscanf()*, *vwscanf()*, or *wscanf()*
 28343 using *stream* that returns data not supplied by a prior call to *ungetwc()*.

28344 **RETURN VALUE**

28345 Upon successful completion, the *fgetwc()* function shall return the wide-character code of the
 28346 character read from the input stream pointed to by *stream* converted to a type **wint_t**. If the end-
 28347 of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for
 28348 the stream shall be set and *fgetwc()* shall return WEOF. If a read error occurs, the error indicator
 28349 CX for the stream shall be set, *fgetwc()* shall return WEOF, and shall set *errno* to indicate the error. If
 28350 an encoding error occurs, the error indicator for the stream shall be set, *fgetwc()* shall return
 28351 WEOF, and shall set *errno* to indicate the error.

28352 **ERRORS**

28353 The *fgetwc()* function shall fail if data needs to be read and:

28354 CX **[EAGAIN]** The **O_NONBLOCK** flag is set for the file descriptor underlying *stream* and
 28355 the thread would be delayed in the *fgetwc()* operation.

28356 CX **[EBADF]** The file descriptor underlying *stream* is not a valid file descriptor open for
 28357 reading.

28358 **[EILSEQ]** The data obtained from the input stream does not form a valid character.

28359 CX **[EINTR]** The read operation was terminated due to the receipt of a signal, and no data
 28360 was transferred.

28361 CX **[EIO]** A physical I/O error has occurred, or the process is in a background process
 28362 group attempting to read from its controlling terminal, and either the process
 28363 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.
 28364 This error may also be generated for implementation-defined reasons.

28365 CX **[EOVERFLOW]** The file is a regular file and an attempt was made to read at or beyond the
 28366 offset maximum associated with the corresponding stream.

28367 The *fgetwc()* function may fail if:

28368 CX [ENOMEM] Insufficient storage space is available.

28369 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
28370 capabilities of the device.

28371 EXAMPLES

28372 None.

28373 APPLICATION USAGE

28374 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
28375 end-of-file condition.

28376 RATIONALE

28377 None.

28378 FUTURE DIRECTIONS

28379 None.

28380 SEE ALSO

28381 *feof()*, *ferror()*, *fopen()*

28382 XBD `<stdio.h>`, `<wchar.h>`

28383 CHANGE HISTORY

28384 First released in Issue 4. Derived from the MSE working draft.

28385 Issue 5

28386 The Optional Header (OH) marking is removed from `<stdio.h>`.

28387 Large File Summit extensions are added.

28388 Issue 6

28389 Extensions beyond the ISO C standard are marked.

28390 The following new requirements on POSIX implementations derive from alignment with the
28391 Single UNIX Specification:

- 28392 • The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- 28393 • The [ENOMEM] and [ENXIO] optional error conditions are added.

28394 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/33 is applied, updating the [EAGAIN]
28395 error in the ERRORS section from “the process would be delayed” to “the thread would be
28396 delayed”.

28397 Issue 7

28398 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

28399 Changes are made related to support for finegrained timestamps.

fgetws()*System Interfaces*28400 **NAME**28401 **fgetws** — get a wide-character string from a stream28402 **SYNOPSIS**28403 `#include <stdio.h>`28404 `#include <wchar.h>`28405 `wchar_t *fgetws(wchar_t *restrict ws, int n,`
28406 `FILE *restrict stream);`28407 **DESCRIPTION**28408 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
28409 conflict between the requirements described here and the ISO C standard is unintentional. This
28410 volume of POSIX.1-2008 defers to the ISO C standard.28411 The *fgetws()* function shall read characters from the *stream*, convert these to the corresponding
28412 wide-character codes, place them in the **wchar_t** array pointed to by *ws*, until *n*–1 characters are
28413 read, or a <newline> is read, converted, and transferred to *ws*, or an end-of-file condition is
28414 encountered. The wide-character string, *ws*, shall then be terminated with a null wide-character
28415 code.

28416 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

28417 **CX** The *fgetws()* function may mark the last data access timestamp of the file associated with *stream*
28418 for update. The last data access timestamp shall be marked for update by the first successful
28419 execution of *fgetwc()*, *fgetws()*, *fwscanf()*, *getwc()*, *getwchar()*, *vfwscanf()*, *vwscanf()*, or *wscanf()*
28420 using *stream* that returns data not supplied by a prior call to *ungetwc()*.28421 **RETURN VALUE**28422 Upon successful completion, *fgetws()* shall return *ws*. If the end-of-file indicator for the stream is
28423 set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and
28424 *fgetws()* shall return a null pointer. If a read error occurs, the error indicator for the stream shall
28425 **CX** be set, *fgetws()* shall return a null pointer, and shall set *errno* to indicate the error.28426 **ERRORS**28427 Refer to *fgetwc()*.28428 **EXAMPLES**

28429 None.

28430 **APPLICATION USAGE**

28431 None.

28432 **RATIONALE**

28433 None.

28434 **FUTURE DIRECTIONS**

28435 None.

28436 **SEE ALSO**28437 *fopen()*, *fread()*28438 XBD `<stdio.h>`, `<wchar.h>`28439 **CHANGE HISTORY**

28440 First released in Issue 4. Derived from the MSE working draft.

- 28441 **Issue 5**
28442 The Optional Header (OH) marking is removed from `<stdio.h>`.
- 28443 **Issue 6**
28444 Extensions beyond the ISO C standard are marked.
28445 The prototype for `fgetws()` is changed for alignment with the ISO/IEC 9899:1999 standard.
- 28446 **Issue 7**
28447 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.
28448 Changes are made related to support for finegrained timestamps.

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

fileno()*System Interfaces*28449 **NAME**28450 **fileno** — map a stream pointer to a file descriptor28451 **SYNOPSIS**

```
28452 CX      #include <stdio.h>
28453      int fileno(FILE *stream);
```

28454 **DESCRIPTION**

28455 The *fileno()* function shall return the integer file descriptor associated with the stream pointed to
 28456 by *stream*.

28457 **RETURN VALUE**

28458 Upon successful completion, *fileno()* shall return the integer value of the file descriptor
 28459 associated with *stream*. Otherwise, the value -1 shall be returned and *errno* set to indicate the
 28460 error.

28461 **ERRORS**28462 The *fileno()* function may fail if:

28463	[EBADF]	The <i>stream</i> argument is not a valid stream, or the stream is not associated with
28464		a file.

28465 **EXAMPLES**

28466 None.

28467 **APPLICATION USAGE**

28468 None.

28469 **RATIONALE**

28470 Without some specification of which file descriptors are associated with these streams, it is
 28471 impossible for an application to set up the streams for another application it starts with *fork()*
 28472 and *exec*. In particular, it would not be possible to write a portable version of the *sh* command
 28473 interpreter (although there may be other constraints that would prevent that portability).

28474 **FUTURE DIRECTIONS**

28475 None.

28476 **SEE ALSO**28477 [Section 2.5.1](#) (on page 491), *dirfd()*, *fdopen()*, *fopen()*, *stdin*28478 XBD [<stdio.h>](#)28479 **CHANGE HISTORY**

28480 First released in Issue 1. Derived from Issue 1 of the SVID.

28481 **Issue 6**

28482 The following new requirements on POSIX implementations derive from alignment with the
 28483 Single UNIX Specification:

- 28484 • The [EBADF] optional error condition is added.

28485 **Issue 7**

28486 SD5-XBD-ERN-99 is applied, changing the definition of the [EBADF] error.

NAME

flockfile, ftrylockfile, funlockfile — stdio locking functions

SYNOPSIS

```
#include <stdio.h>

void flockfile(FILE *file);
int ftrylockfile(FILE *file);
void funlockfile(FILE *file);
```

DESCRIPTION

These functions shall provide for explicit application-level locking of stdio (**FILE ***) objects. These functions can be used by a thread to delineate a sequence of I/O statements that are executed as a unit.

The *flockfile()* function shall acquire for a thread ownership of a (**FILE ***) object.

The *ftrylockfile()* function shall acquire for a thread ownership of a (**FILE ***) object if the object is available; *ftrylockfile()* is a non-blocking version of *flockfile()*.

The *funlockfile()* function shall relinquish the ownership granted to the thread. The behavior is undefined if a thread other than the current owner calls the *funlockfile()* function.

The functions shall behave as if there is a lock count associated with each (**FILE ***) object. This count is implicitly initialized to zero when the (**FILE ***) object is created. The (**FILE ***) object is unlocked when the count is zero. When the count is positive, a single thread owns the (**FILE ***) object. When the *flockfile()* function is called, if the count is zero or if the count is positive and the caller owns the (**FILE ***) object, the count shall be incremented. Otherwise, the calling thread shall be suspended, waiting for the count to return to zero. Each call to *funlockfile()* shall decrement the count. This allows matching calls to *flockfile()* (or successful calls to *ftrylockfile()*) and *funlockfile()* to be nested.

All functions that reference (**FILE ***) objects shall behave as if they use *flockfile()* and *funlockfile()* internally to obtain ownership of these (**FILE ***) objects.

RETURN VALUE

None for *flockfile()* and *funlockfile()*.

The *ftrylockfile()* function shall return zero for success and non-zero to indicate that the lock cannot be acquired.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD [Section 3.285](#) (on page 79).

RATIONALE

The *flockfile()* and *funlockfile()* functions provide an orthogonal mutual-exclusion lock for each **FILE**. The *ftrylockfile()* function provides a non-blocking attempt to acquire a file lock, analogous to *pthread_mutex_trylock()*.

These locks behave as if they are the same as those used internally by *stdio* for thread-safety. This both provides thread-safety of these functions without requiring a second level of internal locking and allows functions in *stdio* to be implemented in terms of other *stdio* functions.

flockfile()*System Interfaces*

28531 Application developers and implementors should be aware that there are potential deadlock
 28532 problems on **FILE** objects. For example, the line-buffered flushing semantics of *stdio* (requested
 28533 via `{_IOLBF}`) require that certain input operations sometimes cause the buffered contents of
 28534 implementation-defined line-buffered output streams to be flushed. If two threads each hold the
 28535 lock on the other's **FILE**, deadlock ensues. This type of deadlock can be avoided by acquiring
 28536 **FILE** locks in a consistent order. In particular, the line-buffered output stream deadlock can
 28537 typically be avoided by acquiring locks on input streams before locks on output streams if a
 28538 thread would be acquiring both.

28539 In summary, threads sharing *stdio* streams with other threads can use *flockfile()* and *funlockfile()*
 28540 to cause sequences of I/O performed by a single thread to be kept bundled. The only case where
 28541 the use of *flockfile()* and *funlockfile()* is required is to provide a scope protecting uses of the
 28542 **_unlocked* functions/macros. This moves the cost/performance tradeoff to the optimal point.

28543 **FUTURE DIRECTIONS**

28544 None.

28545 **SEE ALSO**

28546 *getc_unlocked()*

28547 XBD Section 3.285 (on page 79), `<stdio.h>`

28548 **CHANGE HISTORY**

28549 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

28550 **Issue 6**

28551 These functions are marked as part of the Thread-Safe Functions option.

28552 **Issue 7**

28553 The *flockfile()*, *ftrylockfile()*, and *funlockfile()* functions are moved from the Thread-Safe Functions
 28554 option to the Base.

NAME

floor, floorf, floorl — floor function

SYNOPSIS

```
#include <math.h>

double floor(double x);
float floorf(float x);
long double floorl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall compute the largest integral value not greater than x .

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the largest integral value not greater than x , expressed as a **double**, **float**, or **long double**, as appropriate for the return type of the function.

MX If x is NaN, a NaN shall be returned.

If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

XSI If the correct value would cause overflow, a range error shall occur and *floor()*, *floorf()*, and *floorl()* shall return the value of the macro `-HUGE_VAL`, `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

ERRORS

These functions shall fail if:

XSI **Range Error** The result would cause an overflow.

If the integer expression *(math_errhandling & MATH_ERRNO)* is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression *(math_errhandling & MATH_ERREXCEPT)* is non-zero, then the overflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions might not be expressible as an **int** or **long**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

The *floor()* function can only overflow when the floating-point representation has `DBL_MANT_DIG > DBL_MAX_EXP`.

On error, the expressions *(math_errhandling & MATH_ERRNO)* and *(math_errhandling & MATH_ERREXCEPT)* are independent of each other, but at least one of them must be non-zero.

floor()

System Interfaces

28597 **RATIONALE**

28598 None.

28599 **FUTURE DIRECTIONS**

28600 None.

28601 **SEE ALSO**28602 *ceil()*, *feclearexcept()*, *fetestexcept()*, *isnan()*28603 *Section 4.19* (on page 116), *<math.h>*28604 **CHANGE HISTORY**

28605 First released in Issue 1. Derived from Issue 1 of the SVID.

28606 **Issue 5**28607 The DESCRIPTION is updated to indicate how an application should check for an error. This
28608 text was previously published in the APPLICATION USAGE section.28609 **Issue 6**28610 The *floorf()* and *floorl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.28611 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
28612 revised to align with the ISO/IEC 9899:1999 standard.28613 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
28614 marked.

28615 **NAME**

28616 fma, fmaf, fmal — floating-point multiply-add

28617 **SYNOPSIS**

28618 #include <math.h>

28619 double fma(double x, double y, double z);

28620 float fmaf(float x, float y, float z);

28621 long double fmal(long double x, long double y, long double z);

28622 **DESCRIPTION**

28623 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 28624 conflict between the requirements described here and the ISO C standard is unintentional. This
 28625 volume of POSIX.1-2008 defers to the ISO C standard.

28626 These functions shall compute $(x * y) + z$, rounded as one ternary operation: they shall compute
 28627 the value (as if) to infinite precision and round once to the result format, according to the
 28628 rounding mode characterized by the value of FLT_ROUNDS.

28629 An application wishing to check for error situations should set *errno* to zero and call
 28630 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 28631 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 28632 zero, an error has occurred.

28633 **RETURN VALUE**

28634 Upon successful completion, these functions shall return $(x * y) + z$, rounded as one ternary
 28635 operation.

28636 **MX** If the result overflows or underflows, a range error may occur. On systems that support the IEC
 28637 60559 Floating-Point option, if the result overflows a range error shall occur.

28638 If *x* or *y* are NaN, a NaN shall be returned.

28639 If *x* multiplied by *y* is an exact infinity and *z* is also an infinity but with the opposite sign, a
 28640 domain error shall occur, and either a NaN (if supported), or an implementation-defined value
 28641 shall be returned.

28642 If one of *x* and *y* is infinite, the other is zero, and *z* is not a NaN, a domain error shall occur, and
 28643 either a NaN (if supported), or an implementation-defined value shall be returned.

28644 If one of *x* and *y* is infinite, the other is zero, and *z* is a NaN, a NaN shall be returned and a
 28645 domain error may occur.

28646 If $x*y$ is not $0*\text{Inf}$ nor $\text{Inf}*0$ and *z* is a NaN, a NaN shall be returned.

28647 **ERRORS**

28648 These functions shall fail if:

28649 **MX** **Domain Error** The value of $x*y+z$ is invalid, or the value $x*y$ is invalid and *z* is not a NaN.

28650 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 28651 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 28652 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 28653 shall be raised.

28654 **MX** **Range Error** The result overflows.

28655 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 28656 then *errno* shall be set to [ERANGE]. If the integer expression
 28657 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 28658 floating-point exception shall be raised.

fma()

System Interfaces

28659 These functions may fail if:

28660	MX	Domain Error	The value $x*y$ is invalid and z is a NaN.
28661			If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
28662			then <i>errno</i> shall be set to [EDOM]. If the integer expression (<i>math_errhandling</i>
28663			& MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
28664			shall be raised.
28665		Range Error	The result underflows.
28666			If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
28667			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
28668			(<i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the underflow
28669			floating-point exception shall be raised.
28670		Range Error	The result overflows.
28671			If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
28672			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
28673			(<i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the overflow
28674			floating-point exception shall be raised.

28675 **EXAMPLES**

28676 None.

28677 **APPLICATION USAGE**

28678 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
28679 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

28680 **RATIONALE**

28681 In many cases, clever use of floating (*fused*) multiply-add leads to much improved code; but its
28682 unexpected use by the compiler can undermine carefully written code. The FP_CONTRACT
28683 macro can be used to disallow use of floating multiply-add; and the *fma()* function guarantees
28684 its use where desired. Many current machines provide hardware floating multiply-add
28685 instructions; software implementation can be used for others.

28686 **FUTURE DIRECTIONS**

28687 None.

28688 **SEE ALSO**

28689 *feclearexcept()*, *fecetestexcept()*

28690 XBD Section 4.19 (on page 116), [<math.h>](#)

28691 **CHANGE HISTORY**

28692 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28693 **Issue 7**

28694 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #57 (SD5-XSH-ERN-69) is applied,
28695 adding a “may fail” range error for non-MX systems.

NAME

fmax, fmaxf, fmaxl — determine maximum numeric value of two floating-point numbers

SYNOPSIS

```
#include <math.h>
```

```
double fmax(double x, double y);
float fmaxf(float x, float y);
long double fmaxl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

MX These functions shall determine the maximum numeric value of their arguments. NaN arguments shall be treated as missing data: if one argument is a NaN and the other numeric, then these functions shall choose the numeric value.

RETURN VALUE

Upon successful completion, these functions shall return the maximum numeric value of their arguments.

MX If just one argument is a NaN, the other argument shall be returned.

If *x* and *y* are NaN, a NaN shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fdim(), *fmin()*

XBD *<math.h>*

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #007 is applied.

fmemopen()*System Interfaces*28732 **NAME**28733 **fmemopen** — open a memory buffer stream28734 **SYNOPSIS**

```
28735 CX      #include <stdio.h>
28736
28736 FILE *fmemopen(void *restrict buf, size_t size,
28737                const char *restrict mode);
```

28738 **DESCRIPTION**

28739 The *fmemopen()* function shall associate the buffer given by the *buf* and *size* arguments with a
 28740 stream. The *buf* argument shall be either a null pointer or point to a buffer that is at least *size*
 28741 bytes long.

28742 The *mode* argument is a character string having one of the following values:

28743	<i>r</i> or <i>rb</i>	Open the stream for reading.
28744	<i>w</i> or <i>wb</i>	Open the stream for writing.
28745	<i>a</i> or <i>ab</i>	Append; open the stream for writing at the first null byte.
28746	<i>r+</i> or <i>rb+</i> or <i>r+b</i>	Open the stream for update (reading and writing).
28747	<i>w+</i> or <i>wb+</i> or <i>w+b</i>	Open the stream for update (reading and writing). Truncate the buffer contents.
28749	<i>a+</i> or <i>ab+</i> or <i>a+b</i>	Append; open the stream for update (reading and writing); the initial position is at the first null byte.

28751 The character 'b' shall have no effect.

28752 If a null pointer is specified as the *buf* argument, *fmemopen()* shall allocate *size* bytes of memory
 28753 as if by a call to *malloc()*. This buffer shall be automatically freed when the stream is closed.
 28754 Because this feature is only useful when the stream is opened for updating (because there is no
 28755 way to get a pointer to the buffer) the *fmemopen()* call may fail if the *mode* argument does not
 28756 include a '+'.

28757 The stream maintains a current position in the buffer. This position is initially set to either the
 28758 beginning of the buffer (for *r* and *w* modes) or to the first null byte in the buffer (for *a* modes). If
 28759 no null byte is found in append mode, the initial position is set to one byte after the end of the
 28760 buffer.

28761 If *buf* is a null pointer, the initial position shall always be set to the beginning of the buffer.

28762 The stream also maintains the size of the current buffer contents. For modes *r* and *r+* the size is
 28763 set to the value given by the *size* argument. For modes *w* and *w+* the initial size is zero and for
 28764 modes *a* and *a+* the initial size is either the position of the first null byte in the buffer or the value
 28765 of the *size* argument if no null byte is found.

28766 A read operation on the stream cannot advance the current buffer position beyond the current
 28767 buffer size. Reaching the buffer size in a read operation counts as "end-of-file". Null bytes in the
 28768 buffer have no special meaning for reads. The read operation starts at the current buffer position
 28769 of the stream.

28770 A write operation starts either at the current position of the stream (if mode has not specified
 28771 'a' as the first character) or at the current size of the stream (if mode had 'a' as the first
 28772 character). If the current position at the end of the write is larger than the current buffer size, the
 28773 current buffer size is set to the current position. A write operation on the stream cannot advance
 28774 the current buffer size beyond the size given in the *size* argument.

28775 When a stream open for writing is flushed or closed, a null byte is written at the current position
 28776 or at the end of the buffer, depending on the size of the contents. If a stream open for update is
 28777 flushed or closed and the last write has advanced the current buffer size, a null byte is written at
 28778 the end of the buffer if it fits.

28779 An attempt to seek a memory buffer stream to a negative position or to a position larger than the
 28780 buffer size given in the *size* argument shall fail.

28781 RETURN VALUE

28782 Upon successful completion, *fmemopen()* shall return a pointer to the object controlling the
 28783 stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

28784 ERRORS

28785 The *fmemopen()* function shall fail if:

28786 [EINVAL] The *size* argument specifies a buffer size of zero.

28787 The *fmemopen()* function may fail if:

28788 [EINVAL] The value of the *mode* argument is not valid.

28789 [EINVAL] The *buf* argument is a null pointer and the *mode* argument does not include a
 28790 '+' character.

28791 [ENOMEM] The *buf* argument is a null pointer and the allocation of a buffer of length *size*
 28792 has failed.

28793 [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

28794 EXAMPLES

```
28795 #include <stdio.h>
28796 #include <string.h>
28797 static char buffer[] = "foobar";
28798 int
28799 main (void)
28800 {
28801     int ch;
28802     FILE *stream;
28803     stream = fmemopen(buffer, strlen (buffer), "r");
28804     if (stream == NULL)
28805         /* handle error */;
28806     while ((ch = fgetc(stream)) != EOF)
28807         printf("Got %c\n", ch);
28808     fclose(stream);
28809     return (0);
28810 }
```

28811 This program produces the following output:

```
28812 Got f
28813 Got o
28814 Got o
28815 Got b
28816 Got a
28817 Got r
```

fmemopen()*System Interfaces*28818 **APPLICATION USAGE**

28819 None.

28820 **RATIONALE**

28821 This interface has been introduced to eliminate many of the errors encountered in the
28822 construction of strings, notably overflowing of strings. This interface prevents overflow.

28823 **FUTURE DIRECTIONS**

28824 None.

28825 **SEE ALSO**28826 *fdopen(), fopen(), freopen(), malloc(), open_memstream()*28827 XBD **<stdio.h>**28828 **CHANGE HISTORY**

28829 First released in Issue 7.

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

28830 **NAME**

28831 fmin, fminf, fminl — determine minimum numeric value of two floating-point numbers

28832 **SYNOPSIS**

28833 #include <math.h>

28834 double fmin(double x, double y);

28835 float fminf(float x, float y);

28836 long double fminl(long double x, long double y);

28837 **DESCRIPTION**28838 CX The functionality described on this reference page is aligned with the ISO C standard. Any
28839 conflict between the requirements described here and the ISO C standard is unintentional. This
28840 volume of POSIX.1-2008 defers to the ISO C standard.28841 MX These functions shall determine the minimum numeric value of their arguments. NaN
28842 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,
28843 then these functions shall choose the numeric value.28844 **RETURN VALUE**28845 Upon successful completion, these functions shall return the minimum numeric value of their
28846 arguments.

28847 MX If just one argument is a NaN, the other argument shall be returned.

28848 If *x* and *y* are NaN, a NaN shall be returned.28849 **ERRORS**

28850 No errors are defined.

28851 **EXAMPLES**

28852 None.

28853 **APPLICATION USAGE**

28854 None.

28855 **RATIONALE**

28856 None.

28857 **FUTURE DIRECTIONS**

28858 None.

28859 **SEE ALSO**28860 *fdim()*, *fmax()*

28861 XBD <math.h>

28862 **CHANGE HISTORY**

28863 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28864 **Issue 7**

28865 Austin Group Interpretation 1003.1-2001 #008 is applied.

fmod()*System Interfaces*28866 **NAME**28867 **fmod, fmodf, fmodl** — floating-point remainder value function28868 **SYNOPSIS**28869 `#include <math.h>`28870 `double fmod(double x, double y);`28871 `float fmodf(float x, float y);`28872 `long double fmodl(long double x, long double y);`28873 **DESCRIPTION**

28874 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 28875 conflict between the requirements described here and the ISO C standard is unintentional. This
 28876 volume of POSIX.1-2008 defers to the ISO C standard.

28877 These functions shall return the floating-point remainder of the division of *x* by *y*.

28878 An application wishing to check for error situations should set *errno* to zero and call
 28879 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 28880 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 28881 zero, an error has occurred.

28882 **RETURN VALUE**

28883 These functions shall return the value $x - i * y$, for some integer *i* such that, if *y* is non-zero, the
 28884 result has the same sign as *x* and magnitude less than the magnitude of *y*.

28885 If the correct value would cause underflow, and is not representable, a range error may occur,
 28886 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

28887 **MX** If *x* or *y* is NaN, a NaN shall be returned.

28888 If *y* is zero, a domain error shall occur, and either a NaN (if supported), or an implementation-
 28889 defined value shall be returned.

28890 If *x* is infinite, a domain error shall occur, and either a NaN (if supported), or an
 28891 implementation-defined value shall be returned.

28892 If *x* is ± 0 and *y* is not zero, ± 0 shall be returned.28893 If *x* is not infinite and *y* is $\pm \text{Inf}$, *x* shall be returned.

28894 If the correct value would cause underflow, and is representable, a range error may occur and
 28895 the correct value shall be returned.

28896 **ERRORS**

28897 These functions shall fail if:

28898 **MX** **Domain Error** The *x* argument is infinite or *y* is zero.

28899 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 28900 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 28901 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 28902 shall be raised.

28903 These functions may fail if:

28904 **Range Error** The result underflows.

28905 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 28906 then *errno* shall be set to [ERANGE]. If the integer expression
 28907 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 28908 floating-point exception shall be raised.

28909 EXAMPLES

28910 None.

28911 APPLICATION USAGE

28912 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
28913 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

28914 RATIONALE

28915 None.

28916 FUTURE DIRECTIONS

28917 None.

28918 SEE ALSO

28919 *feclearexcept()*, *fetestexcept()*, *isnan()*

28920 Section 4.19 (on page 116), <math.h>

28921 CHANGE HISTORY

28922 First released in Issue 1. Derived from Issue 1 of the SVID.

28923 Issue 5

28924 The DESCRIPTION is updated to indicate how an application should check for an error. This
28925 text was previously published in the APPLICATION USAGE section.

28926 Issue 6

28927 The behavior for when the *y* argument is zero is now defined.

28928 The *fmodf()* and *fmodl()* functions are added for alignment with the ISO/IEC 9899:1999
28929 standard.

28930 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
28931 revised to align with the ISO/IEC 9899:1999 standard.

28932 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
28933 marked.

fmtmsg()*System Interfaces*28934 **NAME**28935 **fmtmsg** — display a message in the specified format on standard error and/or a system console28936 **SYNOPSIS**

```
28937 XSI      #include <fmtmsg.h>
28938
28938      int fmtmsg(long classification, const char *label, int severity,
28939                const char *text, const char *action, const char *tag);
```

28940 **DESCRIPTION**

28941 The *fmtmsg()* function shall display messages in a specified format instead of the traditional
 28942 *printf()* function.

28943 Based on a message's classification component, *fmtmsg()* shall write a formatted message either
 28944 to standard error, to the console, or to both.

28945 A formatted message consists of up to five components as defined below. The component
 28946 *classification* is not part of a message displayed to the user, but defines the source of the message
 28947 and directs the display of the formatted message.

28948 *classification* Contains the sum of identifying values constructed from the constants defined
 28949 below. Any one identifier from a subclass may be used in combination with a
 28950 single identifier from a different subclass. Two or more identifiers from the
 28951 same subclass should not be used together, with the exception of identifiers
 28952 from the display subclass. (Both display subclass identifiers may be used so
 28953 that messages can be displayed to both standard error and the system
 28954 console.)

28955 **Major Classifications**

28956 Identifies the source of the condition. Identifiers are: MM_HARD
 28957 (hardware), MM_SOFT (software), and MM_FIRM (firmware).

28958 **Message Source Subclassifications**

28959 Identifies the type of software in which the problem is detected.
 28960 Identifiers are: MM_APPL (application), MM_UTIL (utility), and
 28961 MM_OPSYS (operating system).

28962 **Display Subclassifications**

28963 Indicates where the message is to be displayed. Identifiers are:
 28964 MM_PRINT to display the message on the standard error stream,
 28965 MM_CONSOLE to display the message on the system console. One or
 28966 both identifiers may be used.

28967 **Status Subclassifications**

28968 Indicates whether the application can recover from the condition.
 28969 Identifiers are: MM_RECOVER (recoverable) and MM_NRECOV (non-
 28970 recoverable).

28971 An additional identifier, MM_NULLMC, indicates that no classification
 28972 component is supplied for the message.

28973 *label* Identifies the source of the message. The format is two fields separated by a
 28974 <colon>. The first field is up to 10 bytes, the second is up to 14 bytes.

28975 *severity* Indicates the seriousness of the condition. Identifiers for the levels of *severity*
 28976 are:

System Interfaces

fntmsg()

28977		MM_HALT	Indicates that the application has encountered a severe fault and is halting. Produces the string "HALT".
28978			
28979		MM_ERROR	Indicates that the application has detected a fault. Produces the string "ERROR".
28980			
28981		MM_WARNING	Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the string "WARNING".
28982			
28983			
28984		MM_INFO	Provides information about a condition that is not in error. Produces the string "INFO".
28985			
28986		MM_NOSEV	Indicates that no severity level is supplied for the message.
28987	<i>text</i>		Describes the error condition that produced the message. The character string is not limited to a specific size. If the character string is empty, then the text produced is unspecified.
28988			
28989			
28990	<i>action</i>		Describes the first step to be taken in the error-recovery process. The <i>fntmsg()</i> function precedes the action string with the prefix: "TO FIX:". The <i>action</i> string is not limited to a specific size.
28991			
28992			
28993	<i>tag</i>		An identifier that references on-line documentation for the message. Suggested usage is that <i>tag</i> includes the <i>label</i> and a unique identifying number. A sample <i>tag</i> is "XSI:cat:146".
28994			
28995			

28996 The *MSGVERB* environment variable (for message verbosity) shall determine for *fntmsg()*
 28997 which message components it is to select when writing messages to standard error. The value of
 28998 *MSGVERB* shall be a <colon>-separated list of optional keywords. Valid keywords are: *label*,
 28999 *severity*, *text*, *action*, and *tag*. If *MSGVERB* contains a keyword for a component and the
 29000 component's value is not the component's null value, *fntmsg()* shall include that component in
 29001 the message when writing the message to standard error. If *MSGVERB* does not include a
 29002 keyword for a message component, that component shall not be included in the display of the
 29003 message. The keywords may appear in any order. If *MSGVERB* is not defined, if its value is the
 29004 null string, if its value is not of the correct format, or if it contains keywords other than the valid
 29005 ones listed above, *fntmsg()* shall select all components.

29006 *MSGVERB* shall determine which components are selected for display to standard error. All
 29007 message components shall be included in console messages.

RETURN VALUE

29008 The *fntmsg()* function shall return one of the following values:

29010	MM_OK	The function succeeded.
29011	MM_NOTOK	The function failed completely.
29012	MM_NOMSG	The function was unable to generate a message on standard error, but otherwise succeeded.
29013		
29014	MM_NOCON	The function was unable to generate a console message, but otherwise succeeded.
29015		

ERRORS

29016 None.

29017

fmtmsg()*System Interfaces*29018 **EXAMPLES**29019 1. The following example of *fmtmsg()*:

29020 `fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option",`
 29021 `"refer to cat in user's reference manual", "XSI:cat:001")`

29022 produces a complete message in the specified message format:

29023 `XSI:cat: ERROR: illegal option`
 29024 `TO FIX: refer to cat in user's reference manual XSI:cat:001`

29025 2. When the environment variable *MSGVERB* is set as follows:29026 `MSGVERB=severity:text:action`29027 and Example 1 is used, *fmtmsg()* produces:

29028 `ERROR: illegal option`
 29029 `TO FIX: refer to cat in user's reference manual`

29030 **APPLICATION USAGE**

29031 One or more message components may be systematically omitted from messages generated by
 29032 an application by using the null value of the argument for that component.

29033 **RATIONALE**

29034 None.

29035 **FUTURE DIRECTIONS**

29036 None.

29037 **SEE ALSO**29038 *fprintf()*29039 XBD `<fmtmsg.h>`29040 **CHANGE HISTORY**

29041 First released in Issue 4, Version 2.

29042 **Issue 5**

29043 Moved from X/OPEN UNIX extension to BASE.

NAME

fnmatch — match a filename or a pathname

SYNOPSIS

```
#include <fnmatch.h>
```

```
int fnmatch(const char *pattern, const char *string, int flags);
```

DESCRIPTION

The *fnmatch()* function shall match patterns as described in XCU [Section 2.13.1](#) (on page 2332) and [Section 2.13.2](#) (on page 2332). It checks the string specified by the *string* argument to see if it matches the pattern specified by the *pattern* argument.

The *flags* argument shall modify the interpretation of *pattern* and *string*. It is the bitwise-inclusive OR of zero or more of the flags defined in **<fnmatch.h>**. If the FNM_PATHNAME flag is set in *flags*, then a **<slash>** character (**' / '**) in *string* shall be explicitly matched by a **<slash>** in *pattern*; it shall not be matched by either the **<asterisk>** or **<question-mark>** special characters, nor by a bracket expression. If the FNM_PATHNAME flag is not set, the **<slash>** character shall be treated as an ordinary character.

If FNM_NOESCAPE is not set in *flags*, a **<backslash>** character in *pattern* followed by any other character shall match that second character in *string*. In particular, **"\\ "** shall match a **<backslash>** in *string*. If FNM_NOESCAPE is set, a **<backslash>** character shall be treated as an ordinary character.

If FNM_PERIOD is set in *flags*, then a leading **<period>** (**' . '**) in *string* shall match a **<period>** in *pattern*; as described by rule 2 in XCU [Section 2.13.3](#) (on page 2333) where the location of “leading” is indicated by the value of FNM_PATHNAME:

- If FNM_PATHNAME is set, a **<period>** is “leading” if it is the first character in *string* or if it immediately follows a **<slash>**.
- If FNM_PATHNAME is not set, a **<period>** is “leading” only if it is the first character of *string*.

If FNM_PERIOD is not set, then no special restrictions are placed on matching a period.

RETURN VALUE

If *string* matches the pattern specified by *pattern*, then *fnmatch()* shall return 0. If there is no match, *fnmatch()* shall return FNM_NOMATCH, which is defined in **<fnmatch.h>**. If an error occurs, *fnmatch()* shall return another non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *fnmatch()* function has two major uses. It could be used by an application or utility that needs to read a directory and apply a pattern against each entry. The *find* utility is an example of this. It can also be used by the *pax* utility to process its *pattern* operands, or by applications that need to match strings in a similar manner.

The name *fnmatch()* is intended to imply *filename* match, rather than *pathname* match. The default action of this function is to match filenames, rather than pathnames, since it gives no special significance to the **<slash>** character. With the FNM_PATHNAME flag, *fnmatch()* does match pathnames, but without tilde expansion, parameter expansion, or special treatment for a **<period>** at the beginning of a filename.

fnmatch()

System Interfaces

29089 **RATIONALE**

29090 This function replaced the REG_FILENAME flag of *regcomp()* in early proposals of this volume
29091 of POSIX.1-2008. It provides virtually the same functionality as the *regcomp()* and *regexexec()*
29092 functions using the REG_FILENAME and REG_FSLASH flags (the REG_FSLASH flag was
29093 proposed for *regcomp()*, and would have had the opposite effect from FNM_PATHNAME), but
29094 with a simpler function and less system overhead.

29095 **FUTURE DIRECTIONS**

29096 None.

29097 **SEE ALSO**

29098 [*glob\(\)*](#), Section 2.6

29099 XBD [`<fnmatch.h>`](#)

29100 **CHANGE HISTORY**

29101 First released in Issue 4. Derived from the ISO POSIX-2 standard.

29102 **Issue 5**

29103 Moved from POSIX2 C-language Binding to BASE.

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

System Interfaces

fopen()29104 **NAME**29105 `fopen` — open a stream29106 **SYNOPSIS**29107 `#include <stdio.h>`29108 `FILE *fopen(const char *restrict filename, const char *restrict mode);`29109 **DESCRIPTION**

29110 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 29111 conflict between the requirements described here and the ISO C standard is unintentional. This
 29112 volume of POSIX.1-2008 defers to the ISO C standard.

29113 The `fopen()` function shall open the file whose pathname is the string pointed to by `filename`, and
 29114 associates a stream with it.

29115 The `mode` argument points to a string. If the string is one of the following, the file shall be opened
 29116 in the indicated mode. Otherwise, the behavior is undefined.

29117 `r` or `rb` Open file for reading.

29118 `w` or `wb` Truncate to zero length or create file for writing.

29119 `a` or `ab` Append; open or create file for writing at end-of-file.

29120 `r+` or `rb+` or `r+b` Open file for update (reading and writing).

29121 `w+` or `wb+` or `w+b` Truncate to zero length or create file for update.

29122 `a+` or `ab+` or `a+b` Append; open or create file for update, writing at end-of-file.

29123 CX The character '`b`' shall have no effect, but is allowed for ISO C standard conformance. Opening
 29124 a file with read mode (`r` as the first character in the `mode` argument) shall fail if the file does not
 29125 exist or cannot be read.

29126 Opening a file with append mode (`a` as the first character in the `mode` argument) shall cause all
 29127 subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening
 29128 calls to `fseek()`.

29129 When a file is opened with update mode ('`+`' as the second or third character in the `mode`
 29130 argument), both input and output may be performed on the associated stream. However, the
 29131 application shall ensure that output is not directly followed by input without an intervening call
 29132 to `fflush()` or to a file positioning function (`fseek()`, `fsetpos()`, or `rewind()`), and input is not directly
 29133 followed by output without an intervening call to a file positioning function, unless the input
 29134 operation encounters end-of-file.

29135 When opened, a stream is fully buffered if and only if it can be determined not to refer to an
 29136 interactive device. The error and end-of-file indicators for the stream shall be cleared.

29137 CX If `mode` is `w`, `wb`, `a`, `ab`, `w+`, `wb+`, `w+b`, `a+`, `ab+`, or `a+b`, and the file did not previously exist, upon
 29138 successful completion, `fopen()` shall mark for update the last data access, last data modification,
 29139 and last file status change timestamps of the file and the last file status change and last data
 29140 modification timestamps of the parent directory.

29141 If `mode` is `w`, `wb`, `a`, `ab`, `w+`, `wb+`, `w+b`, `a+`, `ab+`, or `a+b`, and the file did not previously exist, the
 29142 `fopen()` function shall create a file as if it called the `creat()` function with a value appropriate for
 29143 the `path` argument interpreted from `filename` and a value of `S_IRUSR` | `S_IWUSR` | `S_IRGRP` |
 29144 `S_IWGRP` | `S_IROTH` | `S_IWOTH` for the `mode` argument.

29145 If `mode` is `w`, `wb`, `w+`, `wb+`, or `w+b`, and the file did previously exist, upon successful completion,
 29146 `fopen()` shall mark for update the last data modification and last file status change timestamps of

fopen()*System Interfaces*

29147 the file.

29148 XSI After a successful call to the *fopen()* function, the orientation of the stream shall be cleared, the
29149 encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an
29150 initial conversion state.

29151 CX The file descriptor associated with the opened stream shall be allocated and opened as if by a
29152 call to *open()* with the following flags:

<i>fopen()</i> Mode	<i>open()</i> Flags
<i>r</i> or <i>rb</i>	O_RDONLY
<i>w</i> or <i>wb</i>	O_WRONLY O_CREAT O_TRUNC
<i>a</i> or <i>ab</i>	O_WRONLY O_CREAT O_APPEND
<i>r+</i> or <i>rb+</i> or <i>r+b</i>	O_RDWR
<i>w+</i> or <i>wb+</i> or <i>w+b</i>	O_RDWR O_CREAT O_TRUNC
<i>a+</i> or <i>ab+</i> or <i>a+b</i>	O_RDWR O_CREAT O_APPEND

RETURN VALUE

29161 Upon successful completion, *fopen()* shall return a pointer to the object controlling the stream.
29162 CX Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

ERRORS

29164 The *fopen()* function shall fail if:

29165 CX [EACCES] Search permission is denied on a component of the path prefix, or the file
29166 exists and the permissions specified by *mode* are denied, or the file does not
29167 exist and write permission is denied for the parent directory of the file to be
29168 created.

29169 CX [EINTR] A signal was caught during *fopen()*.

29170 CX [EISDIR] The named file is a directory and *mode* requires write access.

29171 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
29172 argument.

29173 CX [EMFILE] All file descriptors available to the process are currently open.

29174 CX [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

29175 CX [ENAMETOOLONG]

29176 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
29177 symbolic link produced an intermediate result with a length that exceeds
29178 {PATH_MAX}.

29179 CX [ENFILE] The maximum allowable number of files is currently open in the system.

29180 CX [ENOENT] A component of *filename* does not name an existing file or *filename* is an empty
29181 string.

29182 CX [ENOSPC] The directory or file system that would contain the new file cannot be
29183 expanded, the file does not exist, and the file was to be created.

29184 CX [ENOTDIR] A component of the path prefix is not a directory, or the *filename* argument
29185 contains at least one non-*<slash>* character and ends with one or more trailing
29186 *<slash>* characters and the last pathname component names an existing file
29187 that is neither a directory nor a symbolic link to a directory.

29188	CX	[ENXIO]	The named file is a character special or block special file, and the device associated with this special file does not exist.
29189			
29190	CX	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented correctly in an object of type <code>off_t</code> .
29191			
29192	CX	[EROFS]	The named file resides on a read-only file system and <i>mode</i> requires write access.
29193			
29194			The <i>fopen()</i> function may fail if:
29195	CX	[EINVAL]	The value of the <i>mode</i> argument is not valid.
29196	CX	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
29197			
29198	CX	[EMFILE]	{FOPEN_MAX} streams are currently open in the calling process.
29199	CX	[ENAMETOOLONG]	
29200			The length of a component of a pathname is longer than {NAME_MAX}.
29201	CX	[ENOMEM]	Insufficient storage space is available.
29202	CX	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>mode</i> requires write access.
29203			

EXAMPLES**Opening a File**

The following example tries to open the file named **file** for reading. The *fopen()* function returns a file pointer that is used in subsequent *fgets()* and *fclose()* calls. If the program cannot open the file, it just ignores it.

```
#include <stdio.h>
...
FILE *fp;
...
void rgrep(const char *file)
{
    ...
    if ((fp = fopen(file, "r")) == NULL)
        return;
    ...
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

creat(), *fclose()*, *fdopen()*, *fmemopen()*, *freopen()*, *open_memstream()*

XBD `<stdio.h>`

fopen()*System Interfaces***CHANGE HISTORY**

29229 First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

29231 Large File Summit extensions are added.

Issue 6

29233 Extensions beyond the ISO C standard are marked.

29235 The following new requirements on POSIX implementations derive from alignment with the
29236 Single UNIX Specification:

- 29237 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open
29238 file description. This change is to support large files.
- 29239 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support
29240 large files.
- 29241 • The [ELOOP] mandatory error condition is added.
- 29242 • The [EINVAL], [EMFILE], [ENAMETOOLONG], [ENOMEM], and [ETXTBSY] optional
29243 error conditions are added.

29244 The normative text is updated to avoid use of the term “must” for application requirements.

29245 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 29246 • The prototype for *fopen()* is updated.
- 29247 • The DESCRIPTION is updated to note that if the argument *mode* points to a string other
29248 than those listed, then the behavior is undefined.

29249 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
29250 [ELOOP] error condition is added.

Issue 7

29251 Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying the file creation mode.

29252 Austin Group Interpretation 1003.1-2001 #143 is applied.

29253 Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set
29254 on the open file description.

29255 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

29256 SD5-XSH-ERN-149 is applied, changing the {STREAM_MAX} [EMFILE] error condition from a
29257 “may fail” to a “shall fail”.

29258 Changes are made related to support for finegrained timestamps.

System Interfaces

fork()29260 **NAME**

29261 fork — create a new process

29262 **SYNOPSIS**

29263 #include <unistd.h>

29264 pid_t fork(void);

29265 **DESCRIPTION**

29266 The *fork()* function shall create a new process. The new process (child process) shall be an exact
 29267 copy of the calling process (parent process) except as detailed below:

- 29268 • The child process shall have a unique process ID.
- 29269 • The child process ID also shall not match any active process group ID.
- 29270 • The child process shall have a different parent process ID, which shall be the process ID of
 29271 the calling process.
- 29272 • The child process shall have its own copy of the parent's file descriptors. Each of the
 29273 child's file descriptors shall refer to the same open file description with the corresponding
 29274 file descriptor of the parent.
- 29275 • The child process shall have its own copy of the parent's open directory streams. Each
 29276 open directory stream in the child process may share directory stream positioning with the
 29277 corresponding directory stream of the parent.
- 29278 • The child process shall have its own copy of the parent's message catalog descriptors.
- 29279 • The child process values of *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* shall be set to
 29280 0.
- 29281 • The time left until an alarm clock signal shall be reset to zero, and the alarm, if any, shall be
 29282 canceled; see *alarm()*.
- 29283 XSI • All *semadj* values shall be cleared.
- 29284 • File locks set by the parent process shall not be inherited by the child process.
- 29285 • The set of signals pending for the child process shall be initialized to the empty set.
- 29286 XSI • Interval timers shall be reset in the child process.
- 29287 • Any semaphores that are open in the parent process shall also be open in the child process.
- 29288 ML • The child process shall not inherit any address space memory locks established by the
 29289 parent process via calls to *mlockall()* or *mlock()*.
- 29290 • Memory mappings created in the parent shall be retained in the child process.
 29291 MAP_PRIVATE mappings inherited from the parent shall also be MAP_PRIVATE
 29292 mappings in the child, and any modifications to the data in these mappings made by the
 29293 parent prior to calling *fork()* shall be visible to the child. Any modifications to the data in
 29294 MAP_PRIVATE mappings made by the parent after *fork()* returns shall be visible only to
 29295 the parent. Modifications to the data in MAP_PRIVATE mappings made by the child shall
 29296 be visible only to the child.
- 29297 PS • For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit
 29298 the policy and priority settings of the parent process during a *fork()* function. For other
 29299 scheduling policies, the policy and priority settings on *fork()* are implementation-defined.

fork()*System Interfaces*

- 29300 • Per-process timers created by the parent shall not be inherited by the child process.
- 29301 MSG • The child process shall have its own copy of the message queue descriptors of the parent.
- 29302 Each of the message descriptors of the child shall refer to the same open message queue
- 29303 description as the corresponding message descriptor of the parent.
- 29304 • No asynchronous input or asynchronous output operations shall be inherited by the child
- 29305 process. Any use of asynchronous control blocks created by the parent produces undefined
- 29306 behavior.
- 29307 • A process shall be created with a single thread. If a multi-threaded process calls *fork()*, the
- 29308 new process shall contain a replica of the calling thread and its entire address space,
- 29309 possibly including the states of mutexes and other resources. Consequently, to avoid
- 29310 errors, the child process may only execute async-signal-safe operations until such time as
- 29311 one of the *exec* functions is called. Fork handlers may be established by means of the
- 29312 *pthread_atfork()* function in order to maintain application invariants across *fork()* calls.
- 29313 When the application calls *fork()* from a signal handler and any of the fork handlers
- 29314 registered by *pthread_atfork()* calls a function that is not async-signal-safe, the behavior is
- 29315 undefined.
- 29316 OB TRC TRI • If the Trace option and the Trace Inherit option are both supported:
- 29317 If the calling process was being traced in a trace stream that had its inheritance policy set
- 29318 to *POSIX_TRACE_INHERITED*, the child process shall be traced into that trace stream,
- 29319 and the child process shall inherit the parent's mapping of trace event names to trace event
- 29320 type identifiers. If the trace stream in which the calling process was being traced had its
- 29321 inheritance policy set to *POSIX_TRACE_CLOSE_FOR_CHILD*, the child process shall not
- 29322 be traced into that trace stream. The inheritance policy is set by a call to the
- 29323 *posix_trace_attr_setinherited()* function.
- 29324 OB TRC • If the Trace option is supported, but the Trace Inherit option is not supported:
- 29325 The child process shall not be traced into any of the trace streams of its parent process.
- 29326 OB TRC • If the Trace option is supported, the child process of a trace controller process shall not
- 29327 control the trace streams controlled by its parent process.
- 29328 CPT • The initial value of the CPU-time clock of the child process shall be set to zero.
- 29329 TCT • The initial value of the CPU-time clock of the single thread of the child process shall be set
- 29330 to zero.
- 29331 All other process characteristics defined by POSIX.1-2008 shall be the same in the parent and
- 29332 child processes. The inheritance of process characteristics not defined by POSIX.1-2008 is
- 29333 unspecified by POSIX.1-2008.
- 29334 After *fork()*, both the parent and the child processes shall be capable of executing independently
- 29335 before either one terminates.
- 29336 **RETURN VALUE**
- 29337 Upon successful completion, *fork()* shall return 0 to the child process and shall return the
- 29338 process ID of the child process to the parent process. Both processes shall continue to execute
- 29339 from the *fork()* function. Otherwise, -1 shall be returned to the parent process, no child process
- 29340 shall be created, and *errno* shall be set to indicate the error.

ERRORS

The *fork()* function shall fail if:

[EAGAIN] The system lacked the necessary resources to create another process, or the system-imposed limit on the total number of processes under execution system-wide or by a single user {CHILD_MAX} would be exceeded.

The *fork()* function may fail if:

[ENOMEM] Insufficient storage space is available.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Many historical implementations have timing windows where a signal sent to a process group (for example, an interactive SIGINT) just prior to or during execution of *fork()* is delivered to the parent following the *fork()* but not to the child because the *fork()* code clears the child's set of pending signals. This volume of POSIX.1-2008 does not require, or even permit, this behavior. However, it is pragmatic to expect that problems of this nature may continue to exist in implementations that appear to conform to this volume of POSIX.1-2008 and pass available verification suites. This behavior is only a consequence of the implementation failing to make the interval between signal generation and delivery totally invisible. From the application's perspective, a *fork()* call should appear atomic. A signal that is generated prior to the *fork()* should be delivered prior to the *fork()*. A signal sent to the process group after the *fork()* should be delivered to both parent and child. The implementation may actually initialize internal data structures corresponding to the child's set of pending signals to include signals sent to the process group during the *fork()*. Since the *fork()* call can be considered as atomic from the application's perspective, the set would be initialized as empty and such signals would have arrived after the *fork()*; see also `<signal.h>`.

One approach that has been suggested to address the problem of signal inheritance across *fork()* is to add an [EINTR] error, which would be returned when a signal is detected during the call. While this is preferable to losing signals, it was not considered an optimal solution. Although it is not recommended for this purpose, such an error would be an allowable extension for an implementation.

The [ENOMEM] error value is reserved for those implementations that detect and distinguish such a condition. This condition occurs when an implementation detects that there is not enough memory to create the process. This is intended to be returned when [EAGAIN] is inappropriate because there can never be enough memory (either primary or secondary storage) to perform the operation. Since *fork()* duplicates an existing process, this must be a condition where there is sufficient memory for one such process, but not for two. Many historical implementations actually return [ENOMEM] due to temporary lack of memory, a case that is not generally distinct from [EAGAIN] from the perspective of a conforming application.

Part of the reason for including the optional error [ENOMEM] is because the SVID specifies it and it should be reserved for the error condition specified there. The condition is not applicable on many implementations.

IEEE Std 1003.1-1988 neglected to require concurrent execution of the parent and child of *fork()*. A system that single-threads processes was clearly not intended and is considered an unacceptable "toy implementation" of this volume of POSIX.1-2008. The only objection anticipated to the phrase "executing independently" is testability, but this assertion should be

fork()

testable. Such tests require that both the parent and child can block on a detectable action of the other, such as a write to a pipe or a signal. An interactive exchange of such actions should be possible for the system to conform to the intent of this volume of POSIX.1-2008.

The [EAGAIN] error exists to warn applications that such a condition might occur. Whether it occurs or not is not in any practical sense under the control of the application because the condition is usually a consequence of the user's use of the system, not of the application's code. Thus, no application can or should rely upon its occurrence under any circumstances, nor should the exact semantics of what concept of "user" is used be of concern to the application developer. Validation writers should be cognizant of this limitation.

There are two reasons why POSIX programmers call *fork()*. One reason is to create a new thread of control within the same program (which was originally only possible in POSIX by creating a new process); the other is to create a new process running a different program. In the latter case, the call to *fork()* is soon followed by a call to one of the *exec* functions.

The general problem with making *fork()* work in a multi-threaded world is what to do with all of the threads. There are two alternatives. One is to copy all of the threads into the new process. This causes the programmer or implementation to deal with threads that are suspended on system calls or that might be about to execute system calls that should not be executed in the new process. The other alternative is to copy only the thread that calls *fork()*. This creates the difficulty that the state of process-local resources is usually held in process memory. If a thread that is not calling *fork()* holds a resource, that resource is never released in the child process because the thread whose job it is to release the resource does not exist in the child process.

When a programmer is writing a multi-threaded program, the first described use of *fork()*, creating new threads in the same program, is provided by the *pthread_create()* function. The *fork()* function is thus used only to run new programs, and the effects of calling functions that require certain resources between the call to *fork()* and the call to an *exec* function are undefined.

The addition of the *forkall()* function to the standard was considered and rejected. The *forkall()* function lets all the threads in the parent be duplicated in the child. This essentially duplicates the state of the parent in the child. This allows threads in the child to continue processing and allows locks and the state to be preserved without explicit *pthread_atfork()* code. The calling process has to ensure that the threads processing state that is shared between the parent and child (that is, file descriptors or MAP_SHARED memory) behaves properly after *forkall()*. For example, if a thread is reading a file descriptor in the parent when *forkall()* is called, then two threads (one in the parent and one in the child) are reading the file descriptor after the *forkall()*. If this is not desired behavior, the parent process has to synchronize with such threads before calling *forkall()*.

While the *fork()* function is async-signal-safe, there is no way for an implementation to determine whether the fork handlers established by *pthread_atfork()* are async-signal-safe. The fork handlers may attempt to execute portions of the implementation that are not async-signal-safe, such as those that are protected by mutexes, leading to a deadlock condition. It is therefore undefined for the fork handlers to execute functions that are not async-signal-safe when *fork()* is called from a signal handler.

When *forkall()* is called, threads, other than the calling thread, that are in functions that can return with an [EINTR] error may have those functions return [EINTR] if the implementation cannot ensure that the function behaves correctly in the parent and child. In particular, *pthread_cond_wait()* and *pthread_cond_timedwait()* need to return in order to ensure that the condition has not changed. These functions can be awakened by a spurious condition wakeup rather than returning [EINTR].

FUTURE DIRECTIONS

None.

SEE ALSO

alarm(), *exec*, *fcntl()*, *posix_trace_attr_getinherited()*, *posix_trace_eventid_equal()*, *pthread_atfork()*, *semop()*, *signal()*, *times()*

XBD Section 4.11 (on page 110), **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is changed for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effect of *fork()* on a pending alarm call in the child process is clarified.

The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/17 is applied, adding text to the DESCRIPTION and RATIONALE relating to fork handlers registered by the *pthread_atfork()* function and async-signal safety.

Issue 7

Austin Group Interpretation 1003.1-2001 #080 is applied, clarifying the status of asynchronous input and asynchronous output operations and asynchronous control lists in the DESCRIPTION.

Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Timers, and Threads options is moved to the Base.

Functionality relating to message catalog descriptors is moved from the XSI option to the Base.

fpathconf()

System Interfaces

29465 **NAME**29466 `fpathconf`, `pathconf` — get configurable pathname variables29467 **SYNOPSIS**29468 `#include <unistd.h>`29469 `long fpathconf(int fildes, int name);`29470 `long pathconf(const char *path, int name);`29471 **DESCRIPTION**29472 The `fpathconf()` and `pathconf()` functions shall determine the current value of a configurable limit
29473 or option (*variable*) that is associated with a file or directory.29474 For `pathconf()`, the *path* argument points to the pathname of a file or directory.29475 For `fpathconf()`, the *fildes* argument is an open file descriptor.29476 The *name* argument represents the variable to be queried relative to that file or directory.
29477 Implementations shall support all of the variables listed in the following table and may support
29478 others. The variables in the following table come from `<limits.h>` or `<unistd.h>` and the
29479 symbolic constants, defined in `<unistd.h>`, are the corresponding values used for *name*.

Variable	Value of <i>name</i>	Requirements
{FILESIZEBITS}	_PC_FILESIZEBITS	3,4
{LINK_MAX}	_PC_LINK_MAX	1
{MAX_CANON}	_PC_MAX_CANON	2
{MAX_INPUT}	_PC_MAX_INPUT	2
{NAME_MAX}	_PC_NAME_MAX	3,4
{PATH_MAX}	_PC_PATH_MAX	4,5
{PIPE_BUF}	_PC_PIPE_BUF	6
{POSIX2_SYMLINKS}	_PC_2_SYMLINKS	4
{POSIX_ALLOC_SIZE_MIN}	_PC_ALLOC_SIZE_MIN	10
{POSIX_REC_INCR_XFER_SIZE}	_PC_REC_INCR_XFER_SIZE	10
{POSIX_REC_MAX_XFER_SIZE}	_PC_REC_MAX_XFER_SIZE	10
{POSIX_REC_MIN_XFER_SIZE}	_PC_REC_MIN_XFER_SIZE	10
{POSIX_REC_XFER_ALIGN}	_PC_REC_XFER_ALIGN	10
{SYMLINK_MAX}	_PC_SYMLINK_MAX	4,9
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
_POSIX_NO_TRUNC	_PC_NO_TRUNC	3,4
_POSIX_VDISABLE	_PC_VDISABLE	2
_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
_POSIX_PRIO_IO	_PC_PRIO_IO	8
_POSIX_SYNC_IO	_PC_SYNC_IO	8
_POSIX_TIMESTAMP_RESOLUTION	_PC_TIMESTAMP_RESOLUTION	1

29502 **Requirements**

- 29503 1. If *path* or *fildes* refers to a directory, the value returned shall apply to the directory itself.
- 29504 2. If *path* or *fildes* does not refer to a terminal file, it is unspecified whether an
29505 implementation supports an association of the variable name with the specified file.
- 29506 3. If *path* or *fildes* refers to a directory, the value returned shall apply to filenames within the
29507 directory.

4. If *path* or *filde*s does not refer to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.
5. If *path* or *filde*s refers to a directory, the value returned shall be the maximum length of a relative pathname when the specified directory is the working directory.
6. If *path* refers to a FIFO, or *filde*s refers to a pipe or FIFO, the value returned shall apply to the referenced object. If *path* or *filde*s refers to a directory, the value returned shall apply to any FIFO that exists or can be created within the directory. If *path* or *filde*s refers to any other type of file, it is unspecified whether an implementation supports an association of the variable name with the specified file.
7. If *path* or *filde*s refers to a directory, the value returned shall apply to any files, other than directories, that exist or can be created within the directory.
8. If *path* or *filde*s refers to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.
9. If *path* or *filde*s refers to a directory, the value returned shall be the maximum length of the string that a symbolic link in that directory can contain.
10. If *path* or *filde*s does not refer to a regular file, it is unspecified whether an implementation supports an association of the variable name with the specified file. If an implementation supports such an association for other than a regular file, the value returned is unspecified.

RETURN VALUE

If *name* is an invalid value, both *pathconf()* and *fpathconf()* shall return -1 and set *errno* to indicate the error.

If the variable corresponding to *name* is described in **<limits.h>** as a maximum or minimum value and the variable has no limit for the path or file descriptor, both *pathconf()* and *fpathconf()* shall return -1 without changing *errno*. Note that indefinite limits do not imply infinite limits; see **<limits.h>**.

If the implementation needs to use *path* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *path*, or if the process did not have appropriate privileges to query the file specified by *path*, or *path* does not exist, *pathconf()* shall return -1 and set *errno* to indicate the error.

If the implementation needs to use *filde*s to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *filde*s, or if *filde*s is an invalid file descriptor, *fpathconf()* shall return -1 and set *errno* to indicate the error.

Otherwise, *pathconf()* or *fpathconf()* shall return the current variable value for the file or directory without changing *errno*. The value returned shall not be more restrictive than the corresponding value available to the application when it was compiled with the implementation's **<limits.h>** or **<unistd.h>**.

If the variable corresponding to *name* is dependent on an unsupported option, the results are unspecified.

ERRORS

The *pathconf()* function shall fail if:

[EINVAL] The value of *name* is not valid.

fpathconf()*System Interfaces*

29550	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
29551		
29552	[EOVERFLOW]	The value of <i>name</i> is <code>_PC_TIMESTAMP_RESOLUTION</code> and the resolution is larger than <code>{LONG_MAX}</code> .
29553		
29554	The <i>fpathconf()</i> function may fail if:	
29555	[EACCES]	Search permission is denied for a component of the path prefix.
29556	[EINVAL]	The implementation does not support an association of the variable <i>name</i> with the specified file.
29557		
29558	[ELOOP]	More than <code>{SYMLOOP_MAX}</code> symbolic links were encountered during resolution of the <i>path</i> argument.
29559		
29560	[ENAMETOOLONG]	
29561		The length of a component of a pathname is longer than <code>{NAME_MAX}</code> .
29562	[ENAMETOOLONG]	
29563		The length of a pathname exceeds <code>{PATH_MAX}</code> , or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds <code>{PATH_MAX}</code> .
29564		
29565		
29566	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
29567	[ENOTDIR]	A component of the path prefix is not a directory, or the <i>path</i> argument contains at least one non- <code><slash></code> character and ends with one or more trailing <code><slash></code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
29568		
29569		
29570		
29571	The <i>fpathconf()</i> function shall fail if:	
29572	[EINVAL]	The value of <i>name</i> is not valid.
29573	[EOVERFLOW]	The value of <i>name</i> is <code>_PC_TIMESTAMP_RESOLUTION</code> and the resolution is larger than <code>{LONG_MAX}</code> .
29574		
29575	The <i>fpathconf()</i> function may fail if:	
29576	[EBADF]	The <i>fildev</i> argument is not a valid file descriptor.
29577	[EINVAL]	The implementation does not support an association of the variable <i>name</i> with the specified file.
29578		

EXAMPLES

None.

APPLICATION USAGE

Application developers should check whether an option, such as `_POSIX_ADVISORY_INFO`, is supported prior to obtaining and using values for related variables such as `{POSIX_ALLOC_SIZE_MIN}`.

RATIONALE

The *fpathconf()* function was proposed immediately after the *sysconf()* function when it was realized that some configurable values may differ across file system, directory, or device boundaries.

For example, `{NAME_MAX}` frequently changes between System V and BSD-based file systems; System V uses a maximum of 14, BSD 255. On an implementation that provides both types of file systems, an application would be forced to limit all pathname components to 14 bytes, as this

would be the value specified in `<limits.h>` on such a system.

Therefore, various useful values can be queried on any pathname or file descriptor, assuming that appropriate privileges are in place.

The value returned for the variable `{PATH_MAX}` indicates the longest relative pathname that could be given if the specified directory is the current working directory of the process. A process may not always be able to generate a name that long and use it if a subdirectory in the pathname crosses into a more restrictive file system. Note that implementations are allowed to accept pathnames longer than `{PATH_MAX}` bytes long, but are not allowed to return pathnames longer than this unless the user specifies a larger buffer using a function that provides a buffer size argument.

The value returned for the variable `_POSIX_CHOWN_RESTRICTED` also applies to directories that do not have file systems mounted on them. The value may change when crossing a mount point, so applications that need to know should check for each directory. (An even easier check is to try the `chown()` function and look for an error in case it happens.)

Unlike the values returned by `sysconf()`, the pathname-oriented variables are potentially more volatile and are not guaranteed to remain constant throughout the lifetime of the process. For example, in between two calls to `pathconf()`, the file system in question may have been unmounted and remounted with different characteristics.

Also note that most of the errors are optional. If one of the variables always has the same value on an implementation, the implementation need not look at *path* or *filde*s to return that value and is, therefore, not required to detect any of the errors except the meaning of `[EINVAL]` that indicates that the value of *name* is not valid for that variable.

If the value of any of the limits is unspecified (logically infinite), they will not be defined in `<limits.h>` and the `pathconf()` and `fpathconf()` functions return `-1` without changing `errno`. This can be distinguished from the case of giving an unrecognized *name* argument because `errno` is set to `[EINVAL]` in this case.

Since `-1` is a valid return value for the `pathconf()` and `fpathconf()` functions, applications should set `errno` to zero before calling them and check `errno` only if the return value is `-1`.

For the case of `{SYMLINK_MAX}`, since both `pathconf()` and `open()` follow symbolic links, there is no way that *path* or *filde*s could refer to a symbolic link.

It was the intention of IEEE Std 1003.1d-1999 that the following variables:

```
{POSIX_ALLOC_SIZE_MIN}
{POSIX_REC_INCR_XFER_SIZE}
{POSIX_REC_MAX_XFER_SIZE}
{POSIX_REC_MIN_XFER_SIZE}
{POSIX_REC_XFER_ALIGN}
```

only applied to regular files, but Note 10 also permits implementation of the advisory semantics on other file types unique to an implementation (for example, a character special device).

The `[EOVERFLOW]` error for `_PC_TIMESTAMP_RESOLUTION` cannot occur on POSIX-compliant file systems because POSIX requires a timestamp resolution no larger than one second. Even on 32-bit systems, this can be represented without overflow.

fpathconf()*System Interfaces*29633 **FUTURE DIRECTIONS**

29634 None.

29635 **SEE ALSO**29636 *chown()*, *confstr()*, *sysconf()*29637 XBD *<limits.h>*, *<unistd.h>*29638 XCU *getconf*29639 **CHANGE HISTORY**

29640 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

29641 **Issue 5**

29642 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

29643 Large File Summit extensions are added.

29644 **Issue 6**29645 The following new requirements on POSIX implementations derive from alignment with the
29646 Single UNIX Specification:

- 29647 • The DESCRIPTION is updated to include {FILESIZEBITS}.
- 29648 • The [ELOOP] mandatory error condition is added.
- 29649 • A second [ENAMETOOLONG] is added as an optional error condition.

29650 The following changes were made to align with the IEEE P1003.1a draft standard:

- 29651 • The _PC_SYMLINK_MAX entry is added to the table in the DESCRIPTION.

29652 The following *pathconf()* variables and their associated names are added for alignment with
29653 IEEE Std 1003.1d-1999:

29654 {POSIX_ALLOC_SIZE_MIN}
 29655 {POSIX_REC_INCR_XFER_SIZE}
 29656 {POSIX_REC_MAX_XFER_SIZE}
 29657 {POSIX_REC_MIN_XFER_SIZE}
 29658 {POSIX_REC_XFER_ALIGN}

29659 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/18 is applied, changing the fourth
 29660 paragraph of the DESCRIPTION and removing shading and margin markers from the table.
 29661 This change is needed since implementations are required to support all of these symbols.

29662 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/34 is applied, adding the table entry for
 29663 POSIX2_SYMLINKS in the DESCRIPTION.

29664 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/35 is applied, updating the
 29665 DESCRIPTION and RATIONALE sections to clarify behavior for the following variables:

29666 {POSIX_ALLOC_SIZE_MIN}
 29667 {POSIX_REC_INCR_XFER_SIZE}
 29668 {POSIX_REC_MAX_XFER_SIZE}
 29669 {POSIX_REC_MIN_XFER_SIZE}
 29670 {POSIX_REC_XFER_ALIGN}

29671 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/36 is applied, updating the RETURN
 29672 VALUE and APPLICATION USAGE sections to state that the results are unspecified if a variable
 29673 is dependent on an unsupported option, and advising application developers to check for

29674 supported options prior to obtaining and using such values.

29675 **Issue 7**

29676 Austin Group Interpretations 1003.1-2001 #143 and #160 are applied.

29677 Changes are made related to support for finegrained timestamps.

29678 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
29679 pathname exists but is not a directory or a symbolic link to a directory.

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

fpclassify()

System Interfaces

29680 **NAME**29681 `fpclassify` — classify real floating type29682 **SYNOPSIS**29683 `#include <math.h>`29684 `int fpclassify(real-floating x);`29685 **DESCRIPTION**

29686 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 29687 conflict between the requirements described here and the ISO C standard is unintentional. This
 29688 volume of POSIX.1-2008 defers to the ISO C standard.

29689 The *fpclassify()* macro shall classify its argument value as NaN, infinite, normal, subnormal,
 29690 zero, or into another implementation-defined category. First, an argument represented in a
 29691 format wider than its semantic type is converted to its semantic type. Then classification is based
 29692 on the type of the argument.

29693 **RETURN VALUE**

29694 The *fpclassify()* macro shall return the value of the number classification macro appropriate to
 29695 the value of its argument.

29696 **ERRORS**

29697 No errors are defined.

29698 **EXAMPLES**

29699 None.

29700 **APPLICATION USAGE**

29701 None.

29702 **RATIONALE**

29703 None.

29704 **FUTURE DIRECTIONS**

29705 None.

29706 **SEE ALSO**29707 *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*29708 XBD `<math.h>`29709 **CHANGE HISTORY**

29710 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

NAME

dprintf, fprintf, printf, snprintf, sprintf — print formatted output

SYNOPSIS

```
#include <stdio.h>
```

CX

```
int dprintf(int filides, const char *restrict format, ...);
int fprintf(FILE *restrict stream, const char *restrict format, ...);
int printf(const char *restrict format, ...);
int snprintf(char *restrict s, size_t n,
             const char *restrict format, ...);
int sprintf(char *restrict s, const char *restrict format, ...);
```

DESCRIPTION

CX

Excluding *dprintf()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *fprintf()* function shall place output on the named output *stream*. The *printf()* function shall place output on the standard output stream *stdout*. The *sprintf()* function shall place output followed by the null byte, '`\0`', in consecutive bytes starting at *s*; it is the user's responsibility to ensure that enough space is available.

CX

The *dprintf()* function shall be equivalent to the *fprintf()* function, except that *dprintf()* shall write output to the file associated with the file descriptor specified by the *filides* argument rather than place output on a stream.

The *snprintf()* function shall be equivalent to *sprintf()*, with the addition of the *n* argument which states the size of the buffer referred to by *s*. If *n* is zero, nothing shall be written and *s* may be a null pointer. Otherwise, output bytes beyond the *n*-1st shall be discarded instead of being written to the array, and a null byte is written at the end of the bytes actually written into the array.

If copying takes place between objects that overlap as a result of a call to *sprintf()* or *snprintf()*, the results are undefined.

Each of these functions converts, formats, and prints its arguments under control of the *format*. The *format* is a character string, beginning and ending in its initial shift state, if any. The *format* is composed of zero or more directives: *ordinary characters*, which are simply copied to the output stream, and *conversion specifications*, each of which shall result in the fetching of zero or more arguments. The results are undefined if there are insufficient arguments for the *format*. If the *format* is exhausted while arguments remain, the excess arguments shall be evaluated but are otherwise ignored.

CX

Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier character `%` (see below) is replaced by the sequence "`%n$`", where *n* is a decimal integer in the range `[1, {NL_ARGMAX}]`, giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages (see the EXAMPLES section).

The *format* can contain either numbered argument conversion specifications (that is, "`%n$`" and "`*m$`"), or unnumbered argument conversion specifications (that is, `%` and `*`), but not both. The only exception to this is that `%%` can be mixed with the "`%n$`" form. The results of mixing numbered and unnumbered argument specifications in a *format* string are undefined. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*-1)th, are specified in the format string.

fprintf()*System Interfaces*

- 29758 In format strings containing the "%n\$" form of conversion specification, numbered arguments
29759 in the argument list can be referenced from the format string as many times as required.
- 29760 In format strings containing the % form of conversion specification, each conversion specification
29761 uses the first unused argument in the argument list.
- 29762 CX All forms of the *fprintf()* functions allow for the insertion of a language-dependent radix
29763 character in the output string. The radix character is defined in the process' locale (category
29764 LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the
29765 radix character shall default to a <period> ('.').
- 29766 CX Each conversion specification is introduced by the '%' character or by the character sequence
29767 "%n\$", after which the following appear in sequence:
- 29768 • Zero or more *flags* (in any order), which modify the meaning of the conversion
29769 specification.
 - 29770 • An optional minimum *field width*. If the converted value has fewer bytes than the field
29771 width, it shall be padded with <space> characters by default on the left; it shall be padded
29772 on the right if the left-adjustment flag ('-'), described below, is given to the field width.
29773 The field width takes the form of an <asterisk> ('*'), described below, or a decimal
29774 integer.
 - 29775 • An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u,
29776 x, and X conversion specifiers; the number of digits to appear after the radix character for
29777 the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for
29778 the g and G conversion specifiers; or the maximum number of bytes to be printed from a
29779 XSI string in the s and S conversion specifiers. The precision takes the form of a <period>
29780 ('.') followed either by an <asterisk> ('*'), described below, or an optional decimal digit
29781 string, where a null digit string is treated as zero. If a precision appears with any other
29782 conversion specifier, the behavior is undefined.
 - 29783 • An optional length modifier that specifies the size of the argument.
 - 29784 • A *conversion specifier* character that indicates the type of conversion to be applied.
- 29785 A field width, or precision, or both, may be indicated by an <asterisk> ('*'). In this case an
29786 argument of type **int** supplies the field width or precision. Applications shall ensure that
29787 arguments specifying field width, or precision, or both appear in that order before the argument,
29788 if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field
29789 CX width. A negative precision is taken as if the precision were omitted. In *format* strings
29790 containing the "%n\$" form of a conversion specification, a field width or precision may be
29791 indicated by the sequence "*m\$", where *m* is a decimal integer in the range [1,{NL_ARGMAX}]
29792 giving the position in the argument list (after the *format* argument) of an integer argument
29793 containing the field width or precision, for example:
- 29794 `printf("%1$d:%2$. *3$d:%4$. *3$d\n", hour, min, precision, sec);`
- 29795 The flag characters and their meanings are:
- 29796 CX ' (The <apostrophe>.) The integer portion of the result of a decimal conversion (%i, %d,
29797 %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping characters. For
29798 other conversions the behavior is undefined. The non-monetary grouping character is
29799 used.
- 29800 - The result of the conversion shall be left-justified within the field. The conversion is
29801 right-justified if this flag is not specified.

System Interfaces

fprintf()

29802	+	The result of a signed conversion shall always begin with a sign ('+' or '-'). The
29803		conversion shall begin with a sign only when a negative value is converted if this flag is
29804		not specified.
29805	<space>	If the first character of a signed conversion is not a sign or if a signed conversion results
29806		in no characters, a <space> shall be prefixed to the result. This means that if the
29807		<space> and '+' flags both appear, the <space> flag shall be ignored.
29808	#	Specifies that the value is to be converted to an alternative form. For o conversion, it
29809		increases the precision (if necessary) to force the first digit of the result to be zero. For x
29810		or X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A,
29811		e, E, f, F, g, and G conversion specifiers, the result shall always contain a radix
29812		character, even if no digits follow the radix character. Without this flag, a radix
29813		character appears in the result of these conversions only if a digit follows it. For g and G
29814		conversion specifiers, trailing zeros shall <i>not</i> be removed from the result as they
29815		normally are. For other conversion specifiers, the behavior is undefined.
29816	0	For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros
29817		(following any indication of sign or base) are used to pad to the field width rather than
29818		performing space padding, except when converting an infinity or NaN. If the '0' and
29819		'-' flags both appear, the '0' flag is ignored. For d, i, o, u, x, and X conversion
29820	CX	specifiers, if a precision is specified, the '0' flag shall be ignored. If the '0' and
29821		<apostrophe> flags both appear, the grouping characters are inserted before zero
29822		padding. For other conversions, the behavior is undefined.
29823		The length modifiers and their meanings are:
29824	hh	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a signed char
29825		or unsigned char argument (the argument will have been promoted according to the
29826		integer promotions, but its value shall be converted to signed char or unsigned char
29827		before printing); or that a following n conversion specifier applies to a pointer to a
29828		signed char argument.
29829	h	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a short or
29830		unsigned short argument (the argument will have been promoted according to the
29831		integer promotions, but its value shall be converted to short or unsigned short before
29832		printing); or that a following n conversion specifier applies to a pointer to a short
29833		argument.
29834	l (ell)	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long or
29835		unsigned long argument; that a following n conversion specifier applies to a pointer to
29836		a long argument; that a following c conversion specifier applies to a wint_t argument;
29837		that a following s conversion specifier applies to a pointer to a wchar_t argument; or
29838		has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.
29839	ll (ell-ell)	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long long
29840		or unsigned long long argument; or that a following n conversion specifier applies to a
29841		pointer to a long long argument.
29842		
29843	j	Specifies that a following d, i, o, u, x, or X conversion specifier applies to an intmax_t
29844		or uintmax_t argument; or that a following n conversion specifier applies to a pointer
29845		to an intmax_t argument.
29846	z	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a size_t or the
29847		corresponding signed integer type argument; or that a following n conversion specifier
29848		applies to a pointer to a signed integer type corresponding to a size_t argument.

fprintf()*System Interfaces*

29849	t	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a ptrdiff_t or the corresponding unsigned type argument; or that a following n conversion specifier applies to a pointer to a ptrdiff_t argument.
29850		
29851		
29852	L	Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a long double argument.
29853		
29854		If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.
29855		
29856		The conversion specifiers and their meanings are:
29857	d, i	The int argument shall be converted to a signed decimal in the style "[<i>-</i>] <i>dddd</i> ". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.
29858		
29859		
29860		
29861		
29862	o	The unsigned argument shall be converted to unsigned octal format in the style " <i>dddd</i> ". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.
29863		
29864		
29865		
29866		
29867	u	The unsigned argument shall be converted to unsigned decimal format in the style " <i>dddd</i> ". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.
29868		
29869		
29870		
29871		
29872	x	The unsigned argument shall be converted to unsigned hexadecimal format in the style " <i>dddd</i> "; the letters "abcdef" are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.
29873		
29874		
29875		
29876		
29877	X	Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead of "abcdef".
29878		
29879	f, F	The double argument shall be converted to decimal notation in the style "[<i>-</i>] <i>ddd.d</i> <i>ddd</i> ", where the number of digits after the radix character is equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix character appears, at least one digit appears before it. The low-order digit shall be rounded in an implementation-defined manner.
29880		
29881		
29882		
29883		
29884		
29885		A double argument representing an infinity shall be converted in one of the styles "[<i>-</i>] <i>inf</i> " or "[<i>-</i>] <i>infinity</i> "; which style is implementation-defined. A double argument representing a NaN shall be converted in one of the styles "[<i>-</i>] <i>nan</i> (<i>n-char-sequence</i>)" or "[<i>-</i>] <i>nan</i> "; which style, and the meaning of any <i>n-char-sequence</i> , is implementation-defined. The F conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf", "infinity", or "nan", respectively.
29886		
29887		
29888		
29889		
29890		
29891	e, E	The double argument shall be converted in the style "[<i>-</i>] <i>d.d</i> <i>dde</i> <i>±dd</i> ", where there is one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no radix character shall
29892		
29893		
29894		

29895		appear. The low-order digit shall be rounded in an implementation-defined manner.
29896		The E conversion specifier shall produce a number with 'E' instead of 'e' introducing the exponent. The exponent shall always contain at least two digits. If the
29897		value is zero, the exponent shall be zero.
29898		
29899		A double argument representing an infinity or NaN shall be converted in the style of
29900		an f or F conversion specifier.
29901	g, G	The double argument representing a floating-point number shall be converted in the
29902		style f or e (or in the style F or E in the case of a G conversion specifier), depending on
29903		the value converted and the precision. Let P equal the precision if non-zero, 6 if the
29904		precision is omitted, or 1 if the precision is zero. Then, if a conversion with style E
29905		would have an exponent of X :
29906		— If $P > X \geq -4$, the conversion shall be with style f (or F) and precision $P - (X + 1)$.
29907		— Otherwise, the conversion shall be with style e (or E) and precision $P - 1$.
29908		Finally, unless the '#' flag is used, any trailing zeros shall be removed from the
29909		fractional portion of the result and the decimal-point character shall be removed if there
29910		is no fractional portion remaining.
29911		A double argument representing an infinity or NaN shall be converted in the style of
29912		an f or F conversion specifier.
29913	a, A	A double argument representing a floating-point number shall be converted in the
29914		style "[−]0xh.hhhhp±d", where there is one hexadecimal digit (which shall be non-
29915		zero if the argument is a normalized floating-point number and is otherwise
29916		unspecified) before the decimal-point character and the number of hexadecimal digits
29917		after it is equal to the precision; if the precision is missing and FLT_RADIX is a power
29918		of 2, then the precision shall be sufficient for an exact representation of the value; if the
29919		precision is missing and FLT_RADIX is not a power of 2, then the precision shall be
29920		sufficient to distinguish values of type double , except that trailing zeros may be
29921		omitted; if the precision is zero and the '#' flag is not specified, no decimal-point
29922		character shall appear. The letters "abcdef" shall be used for a conversion and the
29923		letters "ABCDEF" for A conversion. The A conversion specifier produces a number with
29924		'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one
29925		digit, and only as many more digits as necessary to represent the decimal exponent of
29926		2. If the value is zero, the exponent shall be zero.
29927		A double argument representing an infinity or NaN shall be converted in the style of
29928		an f or F conversion specifier.
29929	c	The int argument shall be converted to an unsigned char , and the resulting byte shall
29930		be written.
29931		If an l (ell) qualifier is present, the wint_t argument shall be converted as if by an ls
29932		conversion specification with no precision and an argument that points to a two-
29933		element array of type wchar_t , the first element of which contains the wint_t argument
29934		to the ls conversion specification and the second element contains a null wide
29935		character.
29936	s	The argument shall be a pointer to an array of char . Bytes from the array shall be
29937		written up to (but not including) any terminating null byte. If the precision is specified,
29938		no more than that many bytes shall be written. If the precision is not specified or is
29939		greater than the size of the array, the application shall ensure that the array contains a
29940		null byte.

fprintf()*System Interfaces*

29941			If an l (ell) qualifier is present, the argument shall be a pointer to an array of type
29942			wchar_t . Wide characters from the array shall be converted to characters (each as if by
29943			a call to the <i>wcrtomb()</i> function, with the conversion state described by an mbstate_t
29944			object initialized to zero before the first wide character is converted) up to and
29945			including a terminating null wide character. The resulting characters shall be written
29946			up to (but not including) the terminating null character (byte). If no precision is
29947			specified, the application shall ensure that the array contains a null wide character. If a
29948			precision is specified, no more than that many characters (bytes) shall be written
29949			(including shift sequences, if any), and the array shall contain a null wide character if,
29950			to equal the character sequence length given by the precision, the function would need
29951			to access a wide character one past the end of the array. In no case shall a partial
29952			character be written.
29953	p		The argument shall be a pointer to void . The value of the pointer is converted to a
29954			sequence of printable characters, in an implementation-defined manner.
29955	n		The argument shall be a pointer to an integer into which is written the number of bytes
29956			written to the output so far by this call to one of the <i>fprintf()</i> functions. No argument is
29957			converted.
29958	XSI	C	Equivalent to lc .
29959	XSI	S	Equivalent to ls .
29960	%		Print a ' % ' character; no argument is converted. The complete conversion specification
29961			shall be % %.
29962			If a conversion specification does not match one of the above forms, the behavior is undefined. If
29963			any argument is not the correct type for the corresponding conversion specification, the
29964			behavior is undefined.
29965			In no case shall a nonexistent or small field width cause truncation of a field; if the result of a
29966			conversion is wider than the field width, the field shall be expanded to contain the conversion
29967			result. Characters generated by <i>fprintf()</i> and <i>printf()</i> are printed as if <i>fputc()</i> had been called.
29968			For the a and A conversion specifiers, if FLT_RADIX is a power of 2, the value shall be correctly
29969			rounded to a hexadecimal floating number with the given precision.
29970			For a and A conversions, if FLT_RADIX is not a power of 2 and the result is not exactly
29971			representable in the given precision, the result should be one of the two adjacent numbers in
29972			hexadecimal floating style with the given precision, with the extra stipulation that the error
29973			should have a correct sign for the current rounding direction.
29974			For the e , E , f , F , g , and G conversion specifiers, if the number of significant decimal digits is at
29975			most DECIMAL_DIG , then the result should be correctly rounded. If the number of significant
29976			decimal digits is more than DECIMAL_DIG but the source value is exactly representable with
29977			DECIMAL_DIG digits, then the result should be an exact representation with trailing zeros.
29978			Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having
29979			DECIMAL_DIG significant digits; the value of the resultant decimal string D should satisfy $L \leq$
29980			$D \leq U$, with the extra stipulation that the error should have a correct sign for the current
29981			rounding direction.
29982	CX		The last data modification and last file status change timestamps of the file shall be marked for
29983			update:
29984			1. Between the call to a successful execution of <i>fprintf()</i> or <i>printf()</i> and the next successful
29985			completion of a call to <i>fflush()</i> or <i>fclose()</i> on the same stream or a call to <i>exit()</i> or <i>abort()</i>

2. Upon successful completion of a call to *dprintf()*

RETURN VALUE

Upon successful completion, the *dprintf()*, *fprintf()*, and *printf()* functions shall return the number of bytes transmitted.

Upon successful completion, the *sprintf()* function shall return the number of bytes written to *s*, excluding the terminating null byte.

Upon successful completion, the *snprintf()* function shall return the number of bytes that would be written to *s* had *n* been sufficiently large excluding the terminating null byte.

If an output error was encountered, these functions shall return a negative value and set *errno* to indicate the error.

If the value of *n* is zero on a call to *snprintf()*, nothing shall be written, the number of bytes that would have been written had *n* been sufficiently large excluding the terminating null shall be returned, and *s* may be a null pointer.

ERRORS

For the conditions under which *dprintf()*, *fprintf()*, and *printf()* fail and may fail, refer to *fputc()* or *fputwc()*.

In addition, all forms of *fprintf()* shall fail if:

[EILSEQ] A wide-character code that does not correspond to a valid character has been detected.

In addition, all forms of *fprintf()* may fail if:

[EINVAL] There are insufficient arguments.

The *dprintf()* function may fail if:

[EBADF] The *files* argument is not a valid file descriptor.

The *dprintf()*, *fprintf()*, and *printf()* functions may fail if:

[ENOMEM] Insufficient storage space is available.

The *snprintf()* function shall fail if:

[EOVERFLOW] The value of *n* is greater than {INT_MAX} or the number of bytes needed to hold the output excluding the terminating null is greater than {INT_MAX}.

EXAMPLES

Printing Language-Independent Date and Time

The following statement can be used to print date and time using a language-independent format:

```
printf(format, weekday, month, day, hour, min);
```

For American usage, *format* could be a pointer to the following string:

```
"%s, %s %d, %d:%.2d\n"
```

This example would produce the following message:

```
Sunday, July 3, 10:02
```

For German usage, *format* could be a pointer to the following string:

fprintf()

30024 "%1\$s, %3\$d. %2\$s, %4\$d:%5\$.2d\n"

30025 This definition of *format* would produce the following message:

30026 Sonntag, 3. Juli, 10:02

30027 **Printing File Information**

30028 The following example prints information about the type, permissions, and number of links of a
30029 specific file in a directory.

30030 The first two calls to *printf()* use data decoded from a previous *stat()* call. The user-defined
30031 *strperm()* function shall return a string similar to the one at the beginning of the output for the
30032 following command:

30033 `ls -l`

30034 The next call to *printf()* outputs the owner's name if it is found using *getpwuid()*; the *getpwuid()*
30035 function shall return a **passwd** structure from which the name of the user is extracted. If the user
30036 name is not found, the program instead prints out the numeric value of the user ID.

30037 The next call prints out the group name if it is found using *getgrgid()*; *getgrgid()* is very similar
30038 to *getpwuid()* except that it shall return group information based on the group number. Once
30039 again, if the group is not found, the program prints the numeric value of the group for the entry.

30040 The final call to *printf()* prints the size of the file.

```
30041 #include <stdio.h>
30042 #include <sys/types.h>
30043 #include <pwd.h>
30044 #include <grp.h>

30045 char *strperm (mode_t);
30046 ...
30047 struct stat statbuf;
30048 struct passwd *pwd;
30049 struct group *grp;
30050 ...
30051 printf("%10.10s", strperm (statbuf.st_mode));
30052 printf("%4d", statbuf.st_nlink);

30053 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
30054     printf(" %-8.8s", pwd->pw_name);
30055 else
30056     printf(" %-8ld", (long) statbuf.st_uid);

30057 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
30058     printf(" %-8.8s", grp->gr_name);
30059 else
30060     printf(" %-8ld", (long) statbuf.st_gid);

30061 printf("%9jd", (intmax_t) statbuf.st_size);
30062 ...
```


Printing a Localized Date String

The following example gets a localized date string. The *nl_langinfo()* function shall return the localized date string, which specifies the order and layout of the date. The *strftime()* function takes this information and, using the **tm** structure for values, places the date and time information into *datestring*. The *printf()* function then outputs *datestring* and the name of the entry.

```
#include <stdio.h>
#include <time.h>
#include <langinfo.h>
...
struct dirent *dp;
struct tm *tm;
char datestring[256];
...
strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
printf(" %s %s\n", datestring, dp->d_name);
...
```

Printing Error Information

The following example uses *fprintf()* to write error information to standard error.

In the first group of calls, the program tries to open the password lock file named **LOCKFILE**. If the file already exists, this is an error, as indicated by the **O_EXCL** flag on the *open()* function. If the call fails, the program assumes that someone else is updating the password file, and the program exits.

The next group of calls saves a new password file as the current password file by creating a link between **LOCKFILE** and the new password file **PASSWDFILE**.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

#define LOCKFILE "/etc/ptmp"
#define PASSWDFILE "/etc/passwd"
...
int pfd;
...
if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}
...
if (link(LOCKFILE, PASSWDFILE) == -1) {
```

fprintf()

```

30109         fprintf(stderr, "Link error: %s\n", strerror(errno));
30110         exit(1);
30111     }
30112     ...

```

Printing Usage Information

The following example checks to make sure the program has the necessary arguments, and uses *fprintf()* to print usage information if the expected number of arguments is not present.

```

30116 #include <stdio.h>
30117 #include <stdlib.h>
30118 ...
30119 char *Options = "hdbtl";
30120 ...
30121 if (argc < 2) {
30122     fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
30123 }
30124 ...

```

Formatting a Decimal String

The following example prints a key and data pair on *stdout*. Note use of the <asterisk> ('*') in the format string; this ensures the correct number of decimal places for the element based on the number of elements requested.

```

30129 #include <stdio.h>
30130 ...
30131 long i;
30132 char *keyst;
30133 int elementlen, len;
30134 ...
30135 while (len < elementlen) {
30136     ...
30137     printf("%s Element%0*ld\n", keyst, elementlen, i);
30138     ...
30139 }

```

Creating a Filename

The following example creates a filename using information from a previous *getpwnam()* function that returned the HOME directory of the user.

```

30143 #include <stdio.h>
30144 #include <sys/types.h>
30145 #include <unistd.h>
30146 ...
30147 char filename[PATH_MAX+1];
30148 struct passwd *pw;
30149 ...
30150 sprintf(filename, "%s/%d.out", pw->pw_dir, getpid());
30151 ...

```

Reporting an Event

The following example loops until an event has timed out. The *pause()* function waits forever unless it receives a signal. The *fprintf()* statement should never occur due to the possible return values of *pause()*.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
...
while (!event_complete) {
    ...
    if (pause() != -1 || errno != EINTR)
        fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
}
...
```

Printing Monetary Information

The following example uses *strfmon()* to convert a number and store it as a formatted monetary string named *convbuf*. If the first number is printed, the program prints the format and the description; otherwise, it just prints the number.

```
#include <monetary.h>
#include <stdio.h>
...
struct tblfmt {
    char *format;
    char *description;
};

struct tblfmt table[] = {
    { "%n", "default formatting" },
    { "%11n", "right align within an 11 character field" },
    { "%#5n", "aligned columns for values up to 99 999" },
    { "%=*#5n", "specify a fill character" },
    { "%=0#5n", "fill characters do not use grouping" },
    { "%^#5n", "disable the grouping separator" },
    { "%#5.0n", "round off to whole units" },
    { "%^#5.4n", "increase the precision" },
    { "%(#5n", "use an alternative pos/neg style" },
    { "%!(#5n", "disable the currency symbol" },
};
...
float input[3];
int i, j;
char convbuf[100];
...
strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);

if (j == 0) {
    printf("%s%s%s\n", table[i].format,
        convbuf, table[i].description);
}
```

fprintf()*System Interfaces*

```

30199     }
30200     else {
30201         printf("%s\n", convbuf);
30202     }
30203     ...

```

Printing Wide Characters

The following example prints a series of wide characters. Suppose that "L '@ '" expands to three bytes:

```

30207     wchar_t wz [3] = L"@@";           // Zero-terminated
30208     wchar_t wn [3] = L"@@";           // Unterminated

30209     fprintf (stdout, "%ls", wz);       // Outputs 6 bytes
30210     fprintf (stdout, "%ls", wn);       // Undefined because wn has no terminator
30211     fprintf (stdout, "%4ls", wz);      // Outputs 3 bytes
30212     fprintf (stdout, "%4ls", wn);      // Outputs 3 bytes; no terminator needed
30213     fprintf (stdout, "%9ls", wz);      // Outputs 6 bytes
30214     fprintf (stdout, "%9ls", wn);      // Outputs 9 bytes; no terminator needed
30215     fprintf (stdout, "%10ls", wz);     // Outputs 6 bytes
30216     fprintf (stdout, "%10ls", wn);     // Undefined because wn has no terminator

```

In the last line of the example, after processing three characters, nine bytes have been output. The fourth character must then be examined to determine whether it converts to one byte or more. If it converts to more than one byte, the output is only nine bytes. Since there is no fourth character in the array, the behavior is undefined.

APPLICATION USAGE

If the application calling *fprintf()* has any objects of type **wint_t** or **wchar_t**, it must also include the **<wchar.h>** header to have these objects defined.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fputc(), *fscanf()*, *setlocale()*, *strfmon()*, *wcrtomb()*

XBD Chapter 7 (on page 135), **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the **l** (ell) qualifier can now be used with **c** and **s** conversion specifiers.

The *snprintf()* function is new in Issue 5.

Issue 6

Extensions beyond the ISO C standard are marked.

The normative text is updated to avoid use of the term “must” for application requirements.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The prototypes for *fprintf()*, *printf()*, *snprintf()*, and *sprintf()* are updated, and the XSI shading is removed from *snprintf()*.
- The description of *snprintf()* is aligned with the ISO C standard. Note that this supersedes the *snprintf()* description in The Open Group Base Resolution bwg98-006, which changed the behavior from Issue 5.
- The DESCRIPTION is updated.

The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion specification” consistently.

ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

An example of printing wide characters is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the 0 flag.

Austin Group Interpretation 1003.1-2001 #170 is applied.

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied, revising the description of *g* and *G*.

SD5-XSH-ERN-174 is applied.

The *dprintf()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Functionality relating to the *%n\$* form of conversion specification and the <apostrophe> flag is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

fputc()*System Interfaces*30263 **NAME**30264 **fputc** — put a byte on a stream30265 **SYNOPSIS**30266 `#include <stdio.h>`30267 `int fputc(int c, FILE *stream);`30268 **DESCRIPTION**

30269 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 30270 conflict between the requirements described here and the ISO C standard is unintentional. This
 30271 volume of POSIX.1-2008 defers to the ISO C standard.

30272 The *fputc()* function shall write the byte specified by *c* (converted to an **unsigned char**) to the
 30273 output stream pointed to by *stream*, at the position indicated by the associated file-position
 30274 indicator for the stream (if defined), and shall advance the indicator appropriately. If the file
 30275 cannot support positioning requests, or if the stream was opened with append mode, the byte
 30276 shall be appended to the output stream.

30277 CX The last data modification and last file status change timestamps of the file shall be marked for
 30278 update between the successful execution of *fputc()* and the next successful completion of a call
 30279 to *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

30280 **RETURN VALUE**

30281 Upon successful completion, *fputc()* shall return the value it has written. Otherwise, it shall
 30282 CX return EOF, the error indicator for the stream shall be set, and *errno* shall be set to indicate the
 30283 error.

30284 **ERRORS**

30285 The *fputc()* function shall fail if either the *stream* is unbuffered or the *stream*'s buffer needs to be
 30286 flushed, and:

30287 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and
 30288 the thread would be delayed in the write operation.

30289 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
 30290 writing.

30291 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size.

30292 XSI [EFBIG] An attempt was made to write to a file that exceeds the file size limit of the
 30293 process.

30294 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 30295 offset maximum.

30296 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
 30297 was transferred.

30298 CX [EIO] A physical I/O error has occurred, or the process is a member of a background
 30299 process group attempting to write to its controlling terminal, TOSTOP is set,
 30300 the process is neither ignoring nor blocking SIGTTOU, and the process group
 30301 of the process is orphaned. This error may also be returned under
 30302 implementation-defined conditions.

30303 CX [ENOSPC] There was no free space remaining on the device containing the file.

30304 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 30305 any process. A SIGPIPE signal shall also be sent to the thread.

30306 The *fputc()* function may fail if:

30307 CX [ENOMEM] Insufficient storage space is available.

30308 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
30309 capabilities of the device.

30310 EXAMPLES

30311 None.

30312 APPLICATION USAGE

30313 None.

30314 RATIONALE

30315 None.

30316 FUTURE DIRECTIONS

30317 None.

30318 SEE ALSO

30319 *error()*, *fopen()*, *getrlimit()*, *putc()*, *puts()*, *setbuf()*, *ulimit()*

30320 XBD <stdio.h>

30321 CHANGE HISTORY

30322 First released in Issue 1. Derived from Issue 1 of the SVID.

30323 Issue 5

30324 Large File Summit extensions are added.

30325 Issue 6

30326 Extensions beyond the ISO C standard are marked.

30327 The following new requirements on POSIX implementations derive from alignment with the
30328 Single UNIX Specification:

- 30329 • The [EIO] and [EFBIG] mandatory error conditions are added.
- 30330 • The [ENOMEM] and [ENXIO] optional error conditions are added.

30331 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/37 is applied, updating the [EAGAIN]
30332 error in the ERRORS section from “the process would be delayed” to “the thread would be
30333 delayed”.

30334 Issue 7

30335 Changes are made related to support for finegrained timestamps.

fputs()*System Interfaces*30336 **NAME**

30337 fputs — put a string on a stream

30338 **SYNOPSIS**

30339 #include <stdio.h>

30340 int fputs(const char *restrict *s*, FILE *restrict *stream*);30341 **DESCRIPTION**

30342 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 30343 conflict between the requirements described here and the ISO C standard is unintentional. This
 30344 volume of POSIX.1-2008 defers to the ISO C standard.

30345 The *fputs()* function shall write the null-terminated string pointed to by *s* to the stream pointed
 30346 to by *stream*. The terminating null byte shall not be written.

30347 CX The last data modification and last file status change timestamps of the file shall be marked for
 30348 update between the successful execution of *fputs()* and the next successful completion of a call
 30349 to *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

30350 **RETURN VALUE**

30351 Upon successful completion, *fputs()* shall return a non-negative number. Otherwise, it shall
 30352 CX return EOF, set an error indicator for the stream, and set *errno* to indicate the error.

30353 **ERRORS**30354 Refer to *fputc()*.30355 **EXAMPLES**30356 **Printing to Standard Output**

30357 The following example gets the current time, converts it to a string using *localtime()* and
 30358 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to
 30359 an event for which it is waiting.

```

30360     #include <time.h>
30361     #include <stdio.h>
30362     ...
30363     time_t now;
30364     int minutes_to_event;
30365     ...
30366     time(&now);
30367     printf("The time is ");
30368     fputs(asctime(localtime(&now)), stdout);
30369     printf("There are still %d minutes to the event.\n",
30370           minutes_to_event);
30371     ...
  
```

30372 **APPLICATION USAGE**30373 The *puts()* function appends a <newline> while *fputs()* does not.30374 **RATIONALE**

30375 None.

30376 **FUTURE DIRECTIONS**

30377 None.

30378 **SEE ALSO**30379 *fopen()*, *putc()*, *puts()*30380 XBD *<stdio.h>*30381 **CHANGE HISTORY**

30382 First released in Issue 1. Derived from Issue 1 of the SVID.

30383 **Issue 6**

30384 Extensions beyond the ISO C standard are marked.

30385 The *fputs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.30386 **Issue 7**

30387 Changes are made related to support for finegrained timestamps.

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

fputwc()30388 **NAME**30389 **fputwc** — put a wide-character code on a stream30390 **SYNOPSIS**30391 `#include <stdio.h>`30392 `#include <wchar.h>`30393 `wint_t fputwc(wchar_t wc, FILE *stream);`30394 **DESCRIPTION**

30395 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 30396 conflict between the requirements described here and the ISO C standard is unintentional. This
 30397 volume of POSIX.1-2008 defers to the ISO C standard.

30398 The *fputwc()* function shall write the character corresponding to the wide-character code *wc* to
 30399 the output stream pointed to by *stream*, at the position indicated by the associated file-position
 30400 indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot
 30401 support positioning requests, or if the stream was opened with append mode, the character is
 30402 appended to the output stream. If an error occurs while writing the character, the shift state of
 30403 the output file is left in an undefined state.

30404 CX The last data modification and last file status change timestamps of the file shall be marked for
 30405 update between the successful execution of *fputwc()* and the next successful completion of a call
 30406 to *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

30407 **RETURN VALUE**

30408 Upon successful completion, *fputwc()* shall return *wc*. Otherwise, it shall return WEOF, the error
 30409 CX indicator for the stream shall be set, and *errno* shall be set to indicate the error.

30410 **ERRORS**

30411 The *fputwc()* function shall fail if either the stream is unbuffered or data in the *stream*'s buffer
 30412 needs to be written, and:

30413 CX **[EAGAIN]** The O_NONBLOCK flag is set for the file descriptor underlying *stream* and
 30414 the thread would be delayed in the write operation.

30415 CX **[EBADF]** The file descriptor underlying *stream* is not a valid file descriptor open for
 30416 writing.

30417 CX **[EFBIG]** An attempt was made to write to a file that exceeds the maximum file size or
 30418 the file size limit of the process.

30419 CX **[EFBIG]** The file is a regular file and an attempt was made to write at or beyond the
 30420 offset maximum associated with the corresponding stream.

30421 **[EILSEQ]** The wide-character code *wc* does not correspond to a valid character.

30422 CX **[EINTR]** The write operation was terminated due to the receipt of a signal, and no data
 30423 was transferred.

30424 CX **[EIO]** A physical I/O error has occurred, or the process is a member of a background
 30425 process group attempting to write to its controlling terminal, TOSTOP is set,
 30426 the process is neither ignoring nor blocking SIGTTOU, and the process group
 30427 of the process is orphaned. This error may also be returned under
 30428 implementation-defined conditions.

30429 CX **[ENOSPC]** There was no free space remaining on the device containing the file.

30430 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 30431 any process. A SIGPIPE signal shall also be sent to the thread.

30432 The *fputwc()* function may fail if:

30433 CX [ENOMEM] Insufficient storage space is available.

30434 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 30435 capabilities of the device.

30436 EXAMPLES

30437 None.

30438 APPLICATION USAGE

30439 None.

30440 RATIONALE

30441 None.

30442 FUTURE DIRECTIONS

30443 None.

30444 SEE ALSO

30445 *ferror()*, *fopen()*, *setbuf()*, *ulimit()*

30446 XBD `<stdio.h>`, `<wchar.h>`

30447 CHANGE HISTORY

30448 First released in Issue 4. Derived from the MSE working draft.

30449 Issue 5

30450 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*
 30451 is changed from **wint_t** to **wchar_t**.

30452 The Optional Header (OH) marking is removed from `<stdio.h>`.

30453 Large File Summit extensions are added.

30454 Issue 6

30455 Extensions beyond the ISO C standard are marked.

30456 The following new requirements on POSIX implementations derive from alignment with the
 30457 Single UNIX Specification:

- 30458 • The [EFBIG] and [EIO] mandatory error conditions are added.
- 30459 • The [ENOMEM] and [ENXIO] optional error conditions are added.

30460 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/38 is applied, updating the [EAGAIN]
 30461 error in the ERRORS section from “the process would be delayed” to “the thread would be
 30462 delayed”.

30463 Issue 7

30464 Changes are made related to support for finegrained timestamps.

fputws()*System Interfaces*30465 **NAME**30466 **fputws** — put a wide-character string on a stream30467 **SYNOPSIS**30468 `#include <stdio.h>`30469 `#include <wchar.h>`30470 `int fputws(const wchar_t *restrict ws, FILE *restrict stream);`30471 **DESCRIPTION**

30472 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 30473 conflict between the requirements described here and the ISO C standard is unintentional. This
 30474 volume of POSIX.1-2008 defers to the ISO C standard.

30475 The *fputws()* function shall write a character string corresponding to the (null-terminated) wide-
 30476 character string pointed to by *ws* to the stream pointed to by *stream*. No character corresponding
 30477 to the terminating null wide-character code shall be written.

30478 CX The last data modification and last file status change timestamps of the file shall be marked for
 30479 update between the successful execution of *fputws()* and the next successful completion of a call
 30480 to *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

30481 **RETURN VALUE**

30482 Upon successful completion, *fputws()* shall return a non-negative number. Otherwise, it shall
 30483 CX return -1, set an error indicator for the stream, and set *errno* to indicate the error.

30484 **ERRORS**30485 Refer to *fputwc()*.30486 **EXAMPLES**

30487 None.

30488 **APPLICATION USAGE**30489 The *fputws()* function does not append a <newline>.30490 **RATIONALE**

30491 None.

30492 **FUTURE DIRECTIONS**

30493 None.

30494 **SEE ALSO**30495 *fopen()*

30496 XBD <stdio.h>, <wchar.h>

30497 **CHANGE HISTORY**

30498 First released in Issue 4. Derived from the MSE working draft.

30499 **Issue 5**

30500 The Optional Header (OH) marking is removed from <stdio.h>.

30501 **Issue 6**

30502 Extensions beyond the ISO C standard are marked.

30503 The *fputws()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.30504 **Issue 7**

30505 Changes are made related to support for finegrained timestamps.

System Interfaces

fread()**NAME**

fread — binary input

SYNOPSIS

```
#include <stdio.h>

size_t fread(void *restrict ptr, size_t size, size_t nitems,
FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *fread()* function shall read into the array pointed to by *ptr* up to *nitems* elements whose size is specified by *size* in bytes, from the stream pointed to by *stream*. For each object, *size* calls shall be made to the *fgetc()* function and the results stored, in the order read, in an array of **unsigned char** exactly overlaying the object. The file position indicator for the stream (if defined) shall be advanced by the number of bytes successfully read. If an error occurs, the resulting value of the file position indicator for the stream is unspecified. If a partial element is read, its value is unspecified.

CX The *fread()* function may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*, or *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

RETURN VALUE

Upon successful completion, *fread()* shall return the number of elements successfully read which is less than *nitems* only if a read error or end-of-file is encountered. If *size* or *nitems* is 0, *fread()* shall return 0 and the contents of the array and the state of the stream remain unchanged.

CX Otherwise, if a read error occurs, the error indicator for the stream shall be set, and *errno* shall be set to indicate the error.

ERRORS

Refer to *fgetc()*.

EXAMPLES**Reading from a Stream**

The following example reads a single element from the *fp* stream into the array pointed to by *buf*.

```
#include <stdio.h>
...
size_t bytes_read;
char buf[100];
FILE *fp;
...
bytes_read = fread(buf, sizeof(buf), 1, fp);
...
```

APPLICATION USAGE

The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an end-of-file condition.

Because of possible differences in element length and byte ordering, files written using *fwrite()*

fread()

System Interfaces

30551 are application-dependent, and possibly cannot be read using *fread()* by a different application
30552 or by the same application on a different processor.

RATIONALE

30554 None.

FUTURE DIRECTIONS

30556 None.

SEE ALSO

30558 *feof()*, *ferror()*, *fgetc()*, *fopen()*, *fscanf()*, *getc()*, *gets()*

30559 XBD **<stdio.h>**

CHANGE HISTORY

30561 First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

30563 Extensions beyond the ISO C standard are marked.

30564 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 30565 • The *fread()* prototype is updated.
- 30566 • The DESCRIPTION is updated to describe how the bytes from a call to *fgetc()* are stored.

Issue 7

30568 Changes are made related to support for finegrained timestamps.

NAME

free — free allocated memory

SYNOPSIS

```
#include <stdlib.h>
```

```
void free(void *ptr);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *free()* function shall cause the space pointed to by *ptr* to be deallocated, that is, made available for further allocation. If *ptr* is a null pointer, no action shall occur. Otherwise, if the argument does not match a pointer earlier returned by a function in POSIX.1-2008 that allocates memory as if by *malloc()*, or if the space has been deallocated by a call to *free()* or *realloc()*, the behavior is undefined.

Any use of a pointer that refers to freed space results in undefined behavior.

RETURN VALUE

The *free()* function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

There is now no requirement for the implementation to support the inclusion of **<malloc.h>**.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

calloc(), *malloc()*, *posix_memalign()*, *realloc()*

XBD **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Reference to the *valloc()* function is removed.

Issue 7

The DESCRIPTION is updated to clarify that if the pointer returned is not by a function that allocates memory as if by *malloc()*, then the behavior is undefined.

freeaddrinfo()

System Interfaces

30606 **NAME**

30607 freeaddrinfo, getaddrinfo — get address information

30608 **SYNOPSIS**

```

30609 #include <sys/socket.h>
30610 #include <netdb.h>

30611 void freeaddrinfo(struct addrinfo *ai);
30612 int getaddrinfo(const char *restrict nodename,
30613                const char *restrict servname,
30614                const struct addrinfo *restrict hints,
30615                struct addrinfo **restrict res);

```

30616 **DESCRIPTION**

30617 The *freeaddrinfo()* function shall free one or more **addrinfo** structures returned by *getaddrinfo()*,
 30618 along with any additional storage associated with those structures. If the *ai_next* field of the
 30619 structure is not null, the entire list of structures shall be freed. The *freeaddrinfo()* function shall
 30620 support the freeing of arbitrary sublists of an **addrinfo** list originally returned by *getaddrinfo()*.

30621 The *getaddrinfo()* function shall translate the name of a service location (for example, a host
 30622 name) and/or a service name and shall return a set of socket addresses and associated
 30623 information to be used in creating a socket with which to address the specified service.

30624 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,
 30625 RFC 1035, and RFC 1886.

30626 The *freeaddrinfo()* and *getaddrinfo()* functions shall be thread-safe.

30627 The *nodename* and *servname* arguments are either null pointers or pointers to null-terminated
 30628 strings. One or both of these two arguments shall be supplied by the application as a non-null
 30629 pointer.

30630 The format of a valid name depends on the address family or families. If a specific family is not
 30631 given and the name could be interpreted as valid within multiple supported families, the
 30632 implementation shall attempt to resolve the name in all supported families and, in absence of
 30633 errors, one or more results shall be returned.

30634 If the *nodename* argument is not null, it can be a descriptive name or can be an address string. If
 30635 IP6 the specified address family is AF_INET, AF_INET6, or AF_UNSPEC, valid descriptive names
 30636 include host names. If the specified address family is AF_INET or AF_UNSPEC, address strings
 30637 using Internet standard dot notation as specified in *inet_addr()* are valid.

30638 IP6 If the specified address family is AF_INET6 or AF_UNSPEC, standard IPv6 text forms described
 30639 in *inet_ntop()* are valid.

30640 If *nodename* is not null, the requested service location is named by *nodename*; otherwise, the
 30641 requested service location is local to the caller.

30642 If *servname* is null, the call shall return network-level addresses for the specified *nodename*. If
 30643 *servname* is not null, it is a null-terminated character string identifying the requested service.
 30644 This can be either a descriptive name or a numeric representation suitable for use with the
 30645 IP6 address family or families. If the specified address family is AF_INET, AF_INET6, or
 30646 AF_UNSPEC, the service can be specified as a string specifying a decimal port number.

30647 If the *hints* argument is not null, it refers to a structure containing input values that directs the
 30648 operation by providing options and by limiting the returned information to a specific socket
 30649 type, address family, and/or protocol, as described below. In this *hints* structure every member
 30650 other than *ai_flags*, *ai_family*, *ai_socktype*, and *ai_protocol* shall be set to zero or a null pointer. A
 30651 value of AF_UNSPEC for *ai_family* means that the caller shall accept any address family. A value

of zero for *ai_socktype* means that the caller shall accept any socket type. A value of zero for *ai_protocol* means that the caller shall accept any protocol. If *hints* is a null pointer, the behavior shall be as if it referred to a structure containing the value zero for the *ai_flags*, *ai_socktype*, and *ai_protocol* fields, and AF_UNSPEC for the *ai_family* field.

The *ai_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-inclusive OR of one or more of the values AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST, AI_NUMERICSERV, AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG.

If the AI_PASSIVE flag is specified, the returned address information shall be suitable for use in binding a socket for accepting incoming connections for the specified service. In this case, if the *nodename* argument is null, then the IP address portion of the socket address structure shall be set to INADDR_ANY for an IPv4 address or IN6ADDR_ANY_INIT for an IPv6 address. If the AI_PASSIVE flag is not specified, the returned address information shall be suitable for a call to *connect()* (for a connection-mode protocol) or for a call to *connect()*, *sendto()*, or *sendmsg()* (for a connectionless protocol). In this case, if the *nodename* argument is null, then the IP address portion of the socket address structure shall be set to the loopback address. The AI_PASSIVE flag shall be ignored if the *nodename* argument is not null.

If the AI_CANONNAME flag is specified and the *nodename* argument is not null, the function shall attempt to determine the canonical name corresponding to *nodename* (for example, if *nodename* is an alias or shorthand notation for a complete name).

Note: Since different implementations use different conceptual models, the terms “canonical name” and “alias” cannot be precisely defined for the general case. However, Domain Name System implementations are expected to interpret them as they are used in RFC 1034.

A numeric host address string is not a “name”, and thus does not have a “canonical name” form; no address to host name translation is performed. See below for handling of the case where a canonical name cannot be obtained.

If the AI_NUMERICHOST flag is specified, then a non-null *nodename* string supplied shall be a numeric host address string. Otherwise, an [EAI_NONAME] error is returned. This flag shall prevent any type of name resolution service (for example, the DNS) from being invoked.

If the AI_NUMERICSERV flag is specified, then a non-null *servname* string supplied shall be a numeric port string. Otherwise, an [EAI_NONAME] error shall be returned. This flag shall prevent any type of name resolution service (for example, NIS+) from being invoked.

If the AI_V4MAPPED flag is specified along with an *ai_family* of AF_INET6, then *getaddrinfo()* shall return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses (*ai_addrlen* shall be 16). The AI_V4MAPPED flag shall be ignored unless *ai_family* equals AF_INET6. If the AI_ALL flag is used with the AI_V4MAPPED flag, then *getaddrinfo()* shall return all matching IPv6 and IPv4 addresses. The AI_ALL flag without the AI_V4MAPPED flag is ignored.

If the AI_ADDRCONFIG flag is specified, IPv4 addresses shall be returned only if an IPv4 address is configured on the local system, and IPv6 addresses shall be returned only if an IPv6 address is configured on the local system.

The *ai_socktype* field to which argument *hints* points specifies the socket type for the service, as defined in *socket()*. If a specific socket type is not given (for example, a value of zero) and the service name could be interpreted as valid with multiple supported socket types, the implementation shall attempt to resolve the service name for all supported socket types and, in the absence of errors, all possible results shall be returned. A non-zero socket type value shall limit the returned information to values with the specified socket type.

If the *ai_family* field to which *hints* points has the value AF_UNSPEC, addresses shall be returned for use with any address family that can be used with the specified *nodename* and/or

freeaddrinfo()*System Interfaces*

30699 *servname*. Otherwise, addresses shall be returned for use only with the specified address family.
 30700 If *ai_family* is not AF_UNSPEC and *ai_protocol* is not zero, then addresses shall be returned for
 30701 use only with the specified address family and protocol; the value of *ai_protocol* shall be
 30702 interpreted as in a call to the *socket()* function with the corresponding values of *ai_family* and
 30703 *ai_protocol*.

RETURN VALUE

30704 A zero return value for *getaddrinfo()* indicates successful completion; a non-zero return value
 30705 indicates failure. The possible values for the failures are listed in the ERRORS section.
 30706

30707 Upon successful return of *getaddrinfo()*, the location to which *res* points shall refer to a linked list
 30708 of **addrinfo** structures, each of which shall specify a socket address and information for use in
 30709 creating a socket with which to use that socket address. The list shall include at least one
 30710 **addrinfo** structure. The *ai_next* field of each structure contains a pointer to the next structure on
 30711 the list, or a null pointer if it is the last structure on the list. Each structure on the list shall
 30712 include values for use with a call to the *socket()* function, and a socket address for use with the
 30713 *connect()* function or, if the AI_PASSIVE flag was specified, for use with the *bind()* function. The
 30714 fields *ai_family*, *ai_socktype*, and *ai_protocol* shall be usable as the arguments to the *socket()*
 30715 function to create a socket suitable for use with the returned address. The fields *ai_addr* and
 30716 *ai_addrlen* are usable as the arguments to the *connect()* or *bind()* functions with such a socket,
 30717 according to the AI_PASSIVE flag.

30718 If *nodename* is not null, and if requested by the AI_CANONNAME flag, the *ai_canonname* field of
 30719 the first returned **addrinfo** structure shall point to a null-terminated string containing the
 30720 canonical name corresponding to the input *nodename*; if the canonical name is not available, then
 30721 *ai_canonname* shall refer to the *nodename* argument or a string with the same contents. The
 30722 contents of the *ai_flags* field of the returned structures are undefined.

30723 All fields in socket address structures returned by *getaddrinfo()* that are not filled in through an
 30724 explicit argument (for example, *sin6_flowinfo*) shall be set to zero.

30725 **Note:** This makes it easier to compare socket address structures.

ERRORS

30726 The *getaddrinfo()* function shall fail and return the corresponding error value if:
 30727

30728 [EAI_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

30729 [EAI_BADFLAGS] The *flags* parameter had an invalid value.
 30730

30731 [EAI_FAIL] A non-recoverable error occurred when attempting to resolve the name.

30732 [EAI_FAMILY] The address family was not recognized.

30733 [EAI_MEMORY] There was a memory allocation failure when trying to allocate storage for the
 30734 return value.

30735 [EAI_NONAME] The name does not resolve for the supplied parameters.

30736 Neither *nodename* nor *servname* were supplied. At least one of these shall be
 30737 supplied.

30738 [EAI_SERVICE] The service passed was not recognized for the specified socket type.

30739 [EAI_SOCKTYPE]
 30740 The intended socket type was not recognized.

[EAL_SYSTEM] A system error occurred; the error code can be found in *errno*.

EXAMPLES

The following (incomplete) program demonstrates the use of *getaddrinfo()* to obtain the socket address structure(s) for the service named in the program's command-line argument. The program then loops through each of the address structures attempting to create and bind a socket to the address, until it performs a successful *bind()*.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>

int
main(int argc, char *argv[])
{
    struct addrinfo *result, *rp;
    int sfd, s;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s port\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    struct addrinfo hints = {};
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_flags = AI_PASSIVE;
    hints.ai_protocol = 0;

    s = getaddrinfo(NULL, argv[1], &hints, &result);
    if (s != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
        exit(EXIT_FAILURE);
    }

    /* getaddrinfo() returns a list of address structures.
       Try each address until a successful bind().
       If socket(2) (or bind(2)) fails, close the socket
       and try the next address. */

    for (rp = result; rp != NULL; rp = rp->ai_next) {
        sfd = socket(rp->ai_family, rp->ai_socktype,
                    rp->ai_protocol);
        if (sfd == -1)
            continue;

        if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
            break;          /* Success */

        close(sfd);
    }

    if (rp == NULL) {
        /* No address succeeded */
        fprintf(stderr, "Could not bind\n");
    }
}
```

freeaddrinfo()

System Interfaces

```

30787         exit(EXIT_FAILURE);
30788     }
30789     freeaddrinfo(result);      /* No longer needed */
30790     /* ... use socket bound to sfd ... */
30791 }

```

APPLICATION USAGE

30792 If the caller handles only TCP and not UDP, for example, then the *ai_protocol* member of the *hints*
 30793 structure should be set to IPPROTO_TCP when *getaddrinfo()* is called.
 30794

30795 If the caller handles only IPv4 and not IPv6, then the *ai_family* member of the *hints* structure
 30796 should be set to AF_INET when *getaddrinfo()* is called.

30797 The term “canonical name” is misleading; it is taken from the Domain Name System (RFC 2181).
 30798 It should be noted that the canonical name is a result of alias processing, and not necessarily a
 30799 unique attribute of a host, address, or set of addresses. See RFC 2181 for more discussion of this
 30800 in the Domain Name System context.

RATIONALE

30801 None.
 30802

FUTURE DIRECTIONS

30803 None.
 30804

SEE ALSO

30805 *connect()*, *endservent()*, *gai_strerror()*, *getnameinfo()*, *socket()*
 30806

30807 XBD <netdb.h>, <sys/socket.h>

CHANGE HISTORY

30808 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
 30809

30810 The **restrict** keyword is added to the *getaddrinfo()* prototype for alignment with the
 30811 ISO/IEC 9899:1999 standard.

30812 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/19 is applied, adding three notes to the
 30813 DESCRIPTION and adding text to the APPLICATION USAGE related to the term “canonical
 30814 name”. A reference to RFC 2181 is also added to the Informative References.

30815 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/20 is applied, making changes for
 30816 alignment with IPv6. These include the following:

- 30817 • Adding AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG to the allowed values for the
 30818 *ai_flags* field
- 30819 • Adding a description of AI_ADDRCONFIG
- 30820 • Adding a description of the consequences of ignoring the AI_PASSIVE flag.

30821 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/39 is applied, changing “corresponding
 30822 value” to “corresponding error value” in the ERRORS section.

Issue 7

30823 Austin Group Interpretation 1003.1-2001 #013 is applied.
 30824

30825 Austin Group Interpretation 1003.1-2001 #146 is applied, updating the DESCRIPTION.

30826 An example is added.

NAME

freelocale — free resources allocated for a locale object

SYNOPSIS

```
#include <locale.h>

void freelocale(locale_t locobj);
```

DESCRIPTION

The *freelocale()* function shall cause the resources allocated for a locale object returned by a call to the *newlocale()* or *duplocale()* functions to be released.

Any use of a locale object that has been freed results in undefined behavior.

RETURN VALUE

None.

ERRORS

None.

EXAMPLES**Freeing Up a Locale Object**

The following example shows a code fragment to free a locale object created by *newlocale()*:

```
#include <locale.h>
...

/* Every locale object allocated with newlocale() should be
 * freed using freelocale():
 */

locale_t loc;

/* Get the locale. */

loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);

/* ... Use the locale object ... */
...

/* Free the locale object resources. */
freelocale (loc);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

duplocale(), *newlocale()*, *uselocale()*

XBD [<locale.h>](#)

freelocale()

System Interfaces

30864	CHANGE HISTORY
30865	First released in Issue 7.

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

System Interfaces

freopen()**NAME**

freopen — open a stream

SYNOPSIS

#include <stdio.h>

```
FILE *freopen(const char *restrict filename, const char *restrict mode,
FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *freopen()* function shall first attempt to flush the stream associated with *stream* as if by a call to *fflush(stream)*. Failure to flush the stream successfully shall be ignored. If *filename* is not a null pointer, *freopen()* shall close any file descriptor associated with *stream*. Failure to close the file descriptor successfully shall be ignored. The error and end-of-file indicators for the stream shall be cleared.

The *freopen()* function shall open the file whose pathname is the string pointed to by *filename* and associate the stream pointed to by *stream* with it. The *mode* argument shall be used just as in *fopen()*.

The original stream shall be closed regardless of whether the subsequent open succeeds.

If *filename* is a null pointer, the *freopen()* function shall attempt to change the mode of the stream to that specified by *mode*, as if the name of the file currently associated with the stream had been used. In this case, the file descriptor associated with the stream need not be closed if the call to *freopen()* succeeds. It is implementation-defined which changes of mode are permitted (if any), and under what circumstances.

XSI After a successful call to the *freopen()* function, the orientation of the stream shall be cleared, the encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an initial conversion state.

CX If *filename* is not a null pointer, or if *filename* is a null pointer and the specified mode change necessitates the file descriptor associated with the stream to be closed and reopened, the file descriptor associated with the reopened stream shall be allocated and opened as if by a call to *open()* with the following flags:

<i>freopen()</i> Mode	<i>open()</i> Flags
<i>r</i> or <i>rb</i>	O_RDONLY
<i>w</i> or <i>wb</i>	O_WRONLY O_CREAT O_TRUNC
<i>a</i> or <i>ab</i>	O_WRONLY O_CREAT O_APPEND
<i>r+</i> or <i>rb+</i> or <i>r+b</i>	O_RDWR
<i>w+</i> or <i>wb+</i> or <i>w+b</i>	O_RDWR O_CREAT O_TRUNC
<i>a+</i> or <i>ab+</i> or <i>a+b</i>	O_RDWR O_CREAT O_APPEND

RETURN VALUE

Upon successful completion, *freopen()* shall return the value of *stream*. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

freopen()*System Interfaces***ERRORS**

30907
30908 The *freopen()* function shall fail if:

30909	CX	[EACCES]	Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by <i>mode</i> are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.
30910			
30911			
30912			
30913	CX	[EBADF]	The file descriptor underlying the stream is not a valid file descriptor when <i>filename</i> is a null pointer.
30914			
30915	CX	[EINTR]	A signal was caught during <i>freopen()</i> .
30916	CX	[EISDIR]	The named file is a directory and <i>mode</i> requires write access.
30917	CX	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
30918			
30919	CX	[EMFILE]	All file descriptors available to the process are currently open.
30920	CX	[ENAMETOOLONG]	
30921			The length of a component of a pathname is longer than {NAME_MAX}.
30922	CX	[ENFILE]	The maximum allowable number of files is currently open in the system.
30923	CX	[ENOENT]	A component of <i>filename</i> does not name an existing file or <i>filename</i> is an empty string.
30924			
30925	CX	[ENOSPC]	The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created.
30926			
30927	CX	[ENOTDIR]	A component of the path prefix is not a directory, or the <i>filename</i> argument contains at least one non- <i><slash></i> character and ends with one or more trailing <i><slash></i> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
30928			
30929			
30930			
30931	CX	[ENXIO]	The named file is a character special or block special file, and the device associated with this special file does not exist.
30932			
30933	CX	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented correctly in an object of type <i>off_t</i> .
30934			
30935	CX	[EROFS]	The named file resides on a read-only file system and <i>mode</i> requires write access.
30936			

30937 The *freopen()* function may fail if:

30938	CX	[EBADF]	The mode with which the file descriptor underlying the stream was opened does not support the requested mode when <i>filename</i> is a null pointer.
30939			
30940	CX	[EINVAL]	The value of the <i>mode</i> argument is not valid.
30941	CX	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
30942			
30943	CX	[ENAMETOOLONG]	
30944			The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
30945			
30946			

30947	CX	[ENOMEM]	Insufficient storage space is available.
30948	CX	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
30949			
30950	CX	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>mode</i> requires write access.
30951			

30952 EXAMPLES

30953 Directing Standard Output to a File

30954 The following example logs all standard output to the `/tmp/logfile` file.

```

30955 #include <stdio.h>
30956 ...
30957 FILE *fp;
30958 ...
30959 fp = freopen ("/tmp/logfile", "a+", stdout);
30960 ...

```

30961 APPLICATION USAGE

30962 The `freopen()` function is typically used to attach the pre-opened *streams* associated with *stdin*,
 30963 *stdout*, and *stderr* to other files.

30964 Since implementations are not required to support any stream mode changes when the *filename*
 30965 argument is `NULL`, portable applications cannot rely on the use of `freopen()` to change the stream
 30966 mode, and use of this feature is discouraged. The feature was originally added to the ISO C
 30967 standard in order to facilitate changing *stdin* and *stdout* to binary mode. Since a 'b' character in
 30968 the mode has no effect on POSIX systems, this use of the feature is unnecessary in POSIX
 30969 applications. However, even though the 'b' is ignored, a successful call to `freopen(NULL, "wb",`
 30970 `stdout)` does have an effect. In particular, for regular files it truncates the file and sets the file-
 30971 position indicator for the stream to the start of the file. It is possible that these side-effects are an
 30972 unintended consequence of the way the feature is specified in the ISO/IEC 9899:1999 standard,
 30973 but unless or until the ISO C standard is changed, applications which successfully call
 30974 `freopen(NULL, "wb", stdout)` will behave in unexpected ways on conforming systems in situations
 30975 such as:

```

30976 { appl file1; appl file2; } > file3

```

30977 which will result in `file3` containing only the output from the second invocation of `appl`.

30978 RATIONALE

30979 None.

30980 FUTURE DIRECTIONS

30981 None.

30982 SEE ALSO

30983 `fclose()`, `fdopen()`, `fflush()`, `fmemopen()`, `fopen()`, `mbsinit()`, `open()`, `open_memstream()`

30984 XBD `<stdio.h>`

30985 CHANGE HISTORY

30986 First released in Issue 1. Derived from Issue 1 of the SVID.

freopen()*System Interfaces***Issue 5**

The DESCRIPTION is updated to indicate that the orientation of the stream is cleared and the conversion state of the stream is set to an initial conversion state by a successful call to the *freopen()* function.

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [EINVAL], [ENOMEM], [ENXIO], and [ETXTBSY] optional error conditions are added.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *freopen()* prototype is updated.
- The DESCRIPTION is updated.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

The DESCRIPTION is updated regarding failure to close, changing the “file” to “file descriptor”.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/40 is applied, adding the following sentence to the DESCRIPTION: “In this case, the file descriptor associated with the stream need not be closed if the call to *freopen()* succeeds.”.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/41 is applied, adding an mandatory [EBADF] error, and an optional [EBADF] error to the ERRORS section.

Issue 7

Austin Group Interpretation 1003.1-2001 #043 is applied, clarifying that the *freopen()* function allocates a file descriptor as per *open()*.

Austin Group Interpretation 1003.1-2001 #143 is applied.

Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set on the open file description.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-150 and SD5-XSH-ERN-219 are applied.

31022 **NAME**

31023 frexp, frexpf, frexpl — extract mantissa and exponent from a double precision number

31024 **SYNOPSIS**

```
31025 #include <math.h>
31026 double frexp(double num, int *exp);
31027 float frexpf(float num, int *exp);
31028 long double frexpl(long double num, int *exp);
```

31029 **DESCRIPTION**

31030 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 31031 conflict between the requirements described here and the ISO C standard is unintentional. This
 31032 volume of POSIX.1-2008 defers to the ISO C standard.

31033 These functions shall break a floating-point number *num* into a normalized fraction and an
 31034 integral power of 2. The integer exponent shall be stored in the **int** object pointed to by *exp*.

31035 **RETURN VALUE**

31036 For finite arguments, these functions shall return the value *x*, such that *x* has a magnitude in the
 31037 interval $[\frac{1}{2}, 1)$ or 0, and *num* equals *x* times 2 raised to the power **exp*.

31038 **MX** If *num* is NaN, a NaN shall be returned, and the value of **exp* is unspecified.

31039 If *num* is ± 0 , ± 0 shall be returned, and the value of **exp* shall be 0.

31040 If *num* is $\pm \text{Inf}$, *num* shall be returned, and the value of **exp* is unspecified.

31041 **ERRORS**

31042 No errors are defined.

31043 **EXAMPLES**

31044 None.

31045 **APPLICATION USAGE**

31046 None.

31047 **RATIONALE**

31048 None.

31049 **FUTURE DIRECTIONS**

31050 None.

31051 **SEE ALSO**

31052 *isnan()*, *ldexp()*, *modf()*

31053 XBD *<math.h>*

31054 **CHANGE HISTORY**

31055 First released in Issue 1. Derived from Issue 1 of the SVID.

31056 **Issue 5**

31057 The DESCRIPTION is updated to indicate how an application should check for an error. This
 31058 text was previously published in the APPLICATION USAGE section.

frexp()*System Interfaces*31059 **Issue 6**

31060 The *frexpf()* and *frexpl()* functions are added for alignment with the ISO/IEC 9899:1999
31061 standard.

31062 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
31063 revised to align with the ISO/IEC 9899:1999 standard.

31064 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
31065 marked.

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

System Interfaces

fscanf()**NAME**

`fscanf`, `scanf`, `sscanf` — convert formatted input

SYNOPSIS

```
#include <stdio.h>

int fscanf(FILE *restrict stream, const char *restrict format, ...);
int scanf(const char *restrict format, ...);
int sscanf(const char *restrict s, const char *restrict format, ...);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The `fscanf()` function shall read from the named input *stream*. The `scanf()` function shall read from the standard input stream *stdin*. The `sscanf()` function shall read from the string *s*. Each function reads bytes, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format* described below, and a set of *pointer* arguments indicating where the converted input should be stored. The result is undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments shall be evaluated but otherwise ignored.

Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier character % (see below) is replaced by the sequence "%*n*\$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}]. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages. In format strings containing the "%*n*\$" form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the format string more than once.

The *format* can contain either form of a conversion specification—that is, % or "%*n*\$"—but the two forms cannot be mixed within a single *format* string. The only exception to this is that %% or %* can be mixed with the "%*n*\$" form. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*−1)th, are pointers.

The `fscanf()` function in all its forms shall allow detection of a language-dependent radix character in the input string. The radix character is defined in the locale of the process (category `LC_NUMERIC`). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> ('.').

The *format* is a character string, beginning and ending in its initial shift state, if any, composed of zero or more directives. Each directive is composed of one of the following: one or more white-space characters (<space>, <tab>, <newline>, <vertical-tab>, or <form-feed>); an ordinary character (neither '%' nor a white-space character); or a conversion specification. Each conversion specification is introduced by the character '%' or the character sequence "%*n*\$", after which the following appear in sequence:

- An optional assignment-suppressing character '*'.
- An optional non-zero decimal integer that specifies the maximum field width.
- An optional assignment-allocation character 'm'.
- An option length modifier that specifies the size of the receiving object.

fscanf()*System Interfaces*

- 31110 • A *conversion specifier* character that specifies the type of conversion to be applied. The valid
31111 conversion specifiers are described below.
- 31112 The *fscanf()* functions shall execute each directive of the format in turn. If a directive fails, as
31113 detailed below, the function shall return. Failures are described as input failures (due to the
31114 unavailability of input bytes) or matching failures (due to inappropriate input).
- 31115 A directive composed of one or more white-space characters shall be executed by reading input
31116 until no more valid input can be read, or up to the first byte which is not a white-space character,
31117 which remains unread.
- 31118 A directive that is an ordinary character shall be executed as follows: the next byte shall be read
31119 from the input and compared with the byte that comprises the directive; if the comparison
31120 shows that they are not equivalent, the directive shall fail, and the differing and subsequent
31121 bytes shall remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a
31122 character from being read, the directive shall fail.
- 31123 A directive that is a conversion specification defines a set of matching input sequences, as
31124 described below for each conversion character. A conversion specification shall be executed in
31125 the following steps.
- 31126 Input white-space characters (as specified by *isspace()*) shall be skipped, unless the conversion
31127 specification includes a `[`, `c`, `C`, or `n` conversion specifier.
- 31128 An item shall be read from the input, unless the conversion specification includes an `n`
31129 conversion specifier. An input item shall be defined as the longest sequence of input bytes (up to
31130 any specified maximum field width, which may be measured in characters or bytes dependent
31131 on the conversion specifier) which is an initial subsequence of a matching sequence. The first
31132 byte, if any, after the input item shall remain unread. If the length of the input item is 0, the
31133 execution of the conversion specification shall fail; this condition is a matching failure, unless
31134 end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is
31135 an input failure.
- 31136 Except in the case of a `%` conversion specifier, the input item (or, in the case of a `%n` conversion
31137 specification, the count of input bytes) shall be converted to a type appropriate to the conversion
31138 character. If the input item is not a matching sequence, the execution of the conversion
31139 specification fails; this condition is a matching failure. Unless assignment suppression was
31140 indicated by a `'*'`, the result of the conversion shall be placed in the object pointed to by the
31141 first argument following the *format* argument that has not already received a conversion result if
31142 CX the conversion specification is introduced by `%`, or in the *n*th argument if introduced by the
31143 character sequence `"%n$"`. If this object does not have an appropriate type, or if the result of the
31144 conversion cannot be represented in the space provided, the behavior is undefined.
- 31145 CX The `%c`, `%s`, and `%[` conversion specifiers shall accept an optional assignment-allocation
31146 character `'m'`, which shall cause a memory buffer to be allocated to hold the string converted
31147 including a terminating null character. In such a case, the argument corresponding to the
31148 conversion specifier should be a reference to a pointer variable that will receive a pointer to the
31149 allocated buffer. The system shall allocate a buffer as if *malloc()* had been called. The application
31150 shall be responsible for freeing the memory after usage. If there is insufficient memory to
31151 allocate a buffer, the function shall set *errno* to `[ENOMEM]` and a conversion error shall result. If
31152 the function returns EOF, any memory successfully allocated for parameters using assignment-
31153 allocation character `'m'` by this call shall be freed before the function returns.

31154	The length modifiers and their meanings are:	
31155	hh	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to signed char or unsigned char .
31156		
31157	h	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to short or unsigned short .
31158		
31159	l (ell)	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to long or unsigned long ; that a following a, A, e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to double ; or that a following c, s, or [conversion specifier applies to an argument with type pointer to wchar_t . If the 'm' assignment-allocation character is specified, the conversion applies to an argument with the type pointer to a pointer to wchar_t .
31160		
31161		
31162		
31163	CX	
31164		
31165	ll (ell-ell)	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to long long or unsigned long long .
31166		
31167		
31168	j	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to intmax_t or uintmax_t .
31169		
31170	z	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to size_t or the corresponding signed integer type.
31171		
31172	t	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to ptrdiff_t or the corresponding unsigned type.
31173		
31174	L	Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to long double .
31175		
31176	If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.	
31177		
31178	The following conversion specifiers are valid:	
31179	d	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to int .
31180		
31181		
31182		
31183	i	Matches an optionally signed integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with 0 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to int .
31184		
31185		
31186		
31187	o	Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 8 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
31188		
31189		
31190		
31191	u	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
31192		
31193		
31194		
31195	x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 16 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding
31196		
31197		

fscanf()

System Interfaces

31198		argument is a pointer to unsigned .
31199	a, e, f, g	
31200		Matches an optionally signed floating-point number, infinity, or NaN, whose format is
31201		the same as expected for the subject sequence of <i>strtod</i> (). In the absence of a size
31202		modifier, the application shall ensure that the corresponding argument is a pointer to
31203		float .
31204		If the <i>fprintf</i> () family of functions generates character string representations for infinity
31205		and NaN (a symbolic entity encoded in floating-point format) to support
31206		IEEE Std 754-1985, the <i>fscanf</i> () family of functions shall recognize them as input.
31207	s	Matches a sequence of bytes that are not white-space characters. If the 'm' assignment-
31208		allocation character is not specified, the application shall ensure that the corresponding
31209		argument is a pointer to the initial byte of an array of char , signed char , or unsigned
31210		char large enough to accept the sequence and a terminating null character code, which
31211	CX	shall be added automatically. Otherwise, the application shall ensure that the
31212		corresponding argument is a pointer to a pointer to a char .
31213		If an l (ell) qualifier is present, the input is a sequence of characters that begins in the
31214		initial shift state. Each character shall be converted to a wide character as if by a call to
31215		the <i>mbrtowc</i> () function, with the conversion state described by an mbstate_t object
31216		initialized to zero before the first character is converted. If the 'm' assignment-
31217		allocation character is not specified, the application shall ensure that the corresponding
31218		argument is a pointer to an array of wchar_t large enough to accept the sequence and
31219	CX	the terminating null wide character, which shall be added automatically. Otherwise,
31220		the application shall ensure that the corresponding argument is a pointer to a pointer to
31221		a wchar_t .
31222	[	Matches a non-empty sequence of bytes from a set of expected bytes (the <i>scanset</i>). The
31223		normal skip over white-space characters shall be suppressed in this case. If the 'm'
31224		assignment-allocation character is not specified, the application shall ensure that the
31225		corresponding argument is a pointer to the initial byte of an array of char , signed char ,
31226		or unsigned char large enough to accept the sequence and a terminating null byte,
31227	CX	which shall be added automatically. Otherwise, the application shall ensure that the
31228		corresponding argument is a pointer to a pointer to a char .
31229		If an l (ell) qualifier is present, the input is a sequence of characters that begins in the
31230		initial shift state. Each character in the sequence shall be converted to a wide character
31231		as if by a call to the <i>mbrtowc</i> () function, with the conversion state described by an
31232		mbstate_t object initialized to zero before the first character is converted. If the 'm'
31233		assignment-allocation character is not specified, the application shall ensure that the
31234		corresponding argument is a pointer to an array of wchar_t large enough to accept the
31235	CX	sequence and the terminating null wide character, which shall be added automatically.
31236		Otherwise, the application shall ensure that the corresponding argument is a pointer to
31237		a pointer to a wchar_t .
31238		The conversion specification includes all subsequent bytes in the <i>format</i> string up to
31239		and including the matching <right-square-bracket> (']'). The bytes between the
31240		square brackets (the <i>scanlist</i>) comprise the <i>scanset</i> , unless the byte after the <left-
31241		square-bracket> is a <circumflex> ('^'), in which case the <i>scanset</i> contains all bytes
31242		that do not appear in the <i>scanlist</i> between the <circumflex> and the <right-square-
31243		bracket>. If the conversion specification begins with "[]" or "[^]", the <right-
31244		square-bracket> is included in the <i>scanlist</i> and the next <right-square-bracket> is the
31245		matching <right-square-bracket> that ends the conversion specification; otherwise, the

31246		first <right-square-bracket> is the one that ends the conversion specification. If a '-' is
31247		in the scanlist and is not the first character, nor the second where the first character is a
31248		'^', nor the last character, the behavior is implementation-defined.
31249	c	Matches a sequence of bytes of the number specified by the field width (1 if no field
31250		width is present in the conversion specification). No null byte is added. The normal
31251		skip over white-space characters shall be suppressed in this case. If the 'm'
31252		assignment-allocation character is not specified, the application shall ensure that the
31253		corresponding argument is a pointer to the initial byte of an array of char , signed char ,
31254	CX	or unsigned char large enough to accept the sequence. Otherwise, the application
31255		shall ensure that the corresponding argument is a pointer to a pointer to a char .
31256		If an l (ell) qualifier is present, the input shall be a sequence of characters that begins in
31257		the initial shift state. Each character in the sequence is converted to a wide character as
31258		if by a call to the <i>mbrtowc()</i> function, with the conversion state described by an
31259		mbstate_t object initialized to zero before the first character is converted. No null wide
31260		character is added. If the 'm' assignment-allocation character is not specified, the
31261		application shall ensure that the corresponding argument is a pointer to an array of
31262	CX	wchar_t large enough to accept the resulting sequence of wide characters. Otherwise,
31263		the application shall ensure that the corresponding argument is a pointer to a pointer to
31264		a wchar_t .
31265	p	Matches an implementation-defined set of sequences, which shall be the same as the set
31266		of sequences that is produced by the %p conversion specification of the corresponding
31267		<i>fprintf()</i> functions. The application shall ensure that the corresponding argument is a
31268		pointer to a pointer to void . The interpretation of the input item is implementation-
31269		defined. If the input item is a value converted earlier during the same program
31270		execution, the pointer that results shall compare equal to that value; otherwise, the
31271		behavior of the %p conversion specification is undefined.
31272	n	No input is consumed. The application shall ensure that the corresponding argument is
31273		a pointer to the integer into which shall be written the number of bytes read from the
31274		input so far by this call to the <i>fscanf()</i> functions. Execution of a %n conversion
31275		specification shall not increment the assignment count returned at the completion of
31276		execution of the function. No argument shall be converted, but one shall be consumed.
31277		If the conversion specification includes an assignment-suppressing character or a field
31278		width, the behavior is undefined.
31279	XSI	C Equivalent to l c.
31280	XSI	S Equivalent to l s.
31281	%	Matches a single '%' character; no conversion or assignment occurs. The complete
31282		conversion specification shall be %%.
31283		If a conversion specification is invalid, the behavior is undefined.
31284		The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to a, e, f, g, and
31285		x, respectively.
31286		If end-of-file is encountered during input, conversion shall be terminated. If end-of-file occurs
31287		before any bytes matching the current conversion specification (except for %n) have been read
31288		(other than leading white-space characters, where permitted), execution of the current
31289		conversion specification shall terminate with an input failure. Otherwise, unless execution of the
31290		current conversion specification is terminated with a matching failure, execution of the
31291		following conversion specification (if any) shall be terminated with an input failure.

fscanf()*System Interfaces*

31292 Reaching the end of the string in *sscanf()* shall be equivalent to encountering end-of-file for
31293 *fscanf()*.

31294 If conversion terminates on a conflicting input, the offending input is left unread in the input.
31295 Any trailing white space (including <newline> characters) shall be left unread unless matched
31296 by a conversion specification. The success of literal matches and suppressed assignments is only
31297 directly determinable via the %n conversion specification.

31298 CX The *fscanf()* and *scanf()* functions may mark the last data access timestamp of the file associated
31299 with *stream* for update. The last data access timestamp shall be marked for update by the first
31300 successful execution of *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*,
31301 *fscanf()*, or *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

RETURN VALUE

31302 Upon successful completion, these functions shall return the number of successfully matched
31303 and assigned input items; this number can be zero in the event of an early matching failure. If
31304 the input ends before the first matching failure or conversion, EOF shall be returned. If any
31305 error occurs, EOF shall be returned, and *errno* shall be set to indicate the error. If a read error
31306 CX occurs, the error indicator for the stream shall be set.
31307

ERRORS

31308 For the conditions under which the *fscanf()* functions fail and may fail, refer to *fgetc()* or
31309 *fgetwc()*.
31310

31311 In addition, the *fscanf()* function shall fail if:

31312 CX [EILSEQ] Input byte sequence does not form a valid character.

31313 [ENOMEM] Insufficient storage space is available.

31314 In addition, the *fscanf()* function may fail if:

31315 CX [EINVAL] There are insufficient arguments.

EXAMPLES

31316 The call:

```
31317 int i, n; float x; char name[50];
31318 n = scanf("%d%f%s", &i, &x, name);
```

31319 with the input line:

```
31320 25 54.32E-1 Hamster
```

31321 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string
31322 "Hamster".
31323

31324 The call:

```
31325 int i; float x; char name[50];
31326 (void) scanf("%2d%f%d %[0123456789]", &i, &x, name);
```

31327 with input:

```
31328 56789 0123 56a72
```

31329 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to
31330 *getchar()* shall return the character 'a'.

Reading Data into an Array

The following call uses *fscanf()* to read three floating-point numbers from standard input into the *input* array.

```
float input[3]; fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

APPLICATION USAGE

If the application calling *fscanf()* has any objects of type **wint_t** or **wchar_t**, it must also include the **<wchar.h>** header to have these objects defined.

For functions that allocate memory as if by *malloc()*, the application should release such memory when it is no longer required by a call to *free()*. For *fscanf()*, this is memory allocated via use of the 'm' assignment-allocation character.

RATIONALE

This function is aligned with the ISO/IEC 9899:1999 standard, and in doing so a few “obvious” things were not included. Specifically, the set of characters allowed in a scanset is limited to single-byte characters. In other similar places, multi-byte characters have been permitted, but for alignment with the ISO/IEC 9899:1999 standard, it has not been done here. Applications needing this could use the corresponding wide-character functions to achieve the desired results.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf(), *getc()*, *setlocale()*, *strtod()*, *strtol()*, *strtoul()*, *wcrtomb()*

XBD Chapter 7 (on page 135), **<langinfo.h>**, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the **l** (ell) qualifier is now defined for the **c**, **s**, and **[** conversion specifiers.

The DESCRIPTION is updated to indicate that if infinity and NaN can be generated by the *fprintf()* family of functions, then they are recognized by the *fscanf()* family.

Issue 6

The Open Group Corrigenda U021/7 and U028/10 are applied. These correct several occurrences of “characters” in the text which have been replaced with the term “bytes”.

The normative text is updated to avoid use of the term “must” for application requirements.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The prototypes for *fscanf()*, *scanf()*, and *sscanf()* are updated.
- The DESCRIPTION is updated.
- The **hh**, **ll**, **j**, **t**, and **z** length modifiers are added.
- The **a**, **A**, and **F** conversion characters are added.

The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion specification” consistently.

fscanf()*System Interfaces*31371 **Issue 7**

31372 Austin Group Interpretation 1003.1-2001 #170 is applied.

31373 SD5-XSH-ERN-9 is applied, correcting *fscanf()* to *scanf()* in the DESCRIPTION.

31374 SD5-XSH-ERN-132 is applied, adding the assignment-allocation character 'm'.

31375 Functionality relating to the %n\$ form of conversion specification is moved from the XSI option
31376 to the Base.

31377 Changes are made related to support for finegrained timestamps.

31378 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by
31379 *malloc()*.

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

System Interfaces

fseek()**NAME**

fseek, fseeko — reposition a file-position indicator in a stream

SYNOPSIS

```
#include <stdio.h>
```

```
int fseek(FILE *stream, long offset, int whence);
```

```
CX int fseeko(FILE *stream, off_t offset, int whence);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *fseek()* function shall set the file-position indicator for the stream pointed to by *stream*. If a read or write error occurs, the error indicator for the stream shall be set and *fseek()* fails.

The new position, measured in bytes from the beginning of the file, shall be obtained by adding *offset* to the position specified by *whence*. The specified point is the beginning of the file for *SEEK_SET*, the current value of the file-position indicator for *SEEK_CUR*, or end-of-file for *SEEK_END*.

If the stream is to be used with wide-character input/output functions, the application shall ensure that *offset* is either 0 or a value returned by an earlier call to *ftell()* on the same stream and *whence* is *SEEK_SET*.

A successful call to *fseek()* shall clear the end-of-file indicator for the stream and undo any effects of *ungetc()* and *ungetwc()* on the same stream. After an *fseek()* call, the next operation on an update stream may be either input or output.

CX If the most recent operation, other than *ftell()*, on a given stream is *fflush()*, the file offset in the underlying open file description shall be adjusted to reflect the location specified by *fseek()*.

The *fseek()* function shall allow the file-position indicator to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads of data in the gap shall return bytes with the value 0 until data is actually written into the gap.

The behavior of *fseek()* on devices which are incapable of seeking is implementation-defined. The value of the file offset associated with such a device is undefined.

If the stream is writable and buffered data had not been written to the underlying file, *fseek()* shall cause the unwritten data to be written to the file and shall mark the last data modification and last file status change timestamps of the file for update.

In a locale with state-dependent encoding, whether *fseek()* restores the stream's shift state is implementation-defined.

The *fseeko()* function shall be equivalent to the *fseek()* function except that the *offset* argument is of type *off_t*.

RETURN VALUE

CX The *fseek()* and *fseeko()* functions shall return 0 if they succeed.

CX Otherwise, they shall return -1 and set *errno* to indicate the error.

ERRORS

CX The *fseek()* and *fseeko()* functions shall fail if, either the *stream* is unbuffered or the *stream's* buffer needed to be flushed, and the call to *fseek()* or *fseeko()* causes an underlying *lseek()* or *write()* to be invoked, and:

fseek()*System Interfaces*

31423	CX	[EAGAIN]	The O_NONBLOCK flag is set for the file descriptor and the thread would be delayed in the write operation.
31424			
31425	CX	[EBADF]	The file descriptor underlying the stream file is not open for writing or the stream's buffer needed to be flushed and the file is not open.
31426			
31427	CX	[EFBIG]	An attempt was made to write a file that exceeds the maximum file size.
31428	XSI	[EFBIG]	An attempt was made to write a file that exceeds the file size limit of the process.
31429			
31430	CX	[EFBIG]	The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.
31431			
31432	CX	[EINTR]	The write operation was terminated due to the receipt of a signal, and no data was transferred.
31433			
31434	CX	[EINVAL]	The <i>whence</i> argument is invalid. The resulting file-position indicator would be set to a negative value.
31435			
31436	CX	[EIO]	A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a <i>write()</i> to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
31437			
31438			
31439			
31440			
31441	CX	[ENOSPC]	There was no free space remaining on the device containing the file.
31442	CX	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
31443			
31444	CX	[EOVERFLOW]	For <i>fseek()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type long .
31445			
31446	CX	[EOVERFLOW]	For <i>fseeko()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type off_t .
31447			
31448	CX	[EPIPE]	An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a SIGPIPE signal shall also be sent to the thread.
31449			
31450	CX	[ESPIPE]	The file descriptor underlying <i>stream</i> is associated with a pipe or FIFO.

EXAMPLES

31451
31452 None.

APPLICATION USAGE

31453
31454 None.

RATIONALE

31455
31456 None.

FUTURE DIRECTIONS

31457
31458 None.

SEE ALSO

31459 *fopen()*, *fsetpos()*, *ftell()*, *getrlimit()*, *lseek()*, *rewind()*, *ulimit()*, *ungetc()*, *write()*

31460
31461 XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *fseeko()* function is added.
- The [EFBIG], [EOVERFLOW], and [ENXIO] mandatory error conditions are added.

The following change is incorporated for alignment with the FIPS requirements:

- The [EINTR] error is no longer an indication that the implementation does not report partial transfers.

The normative text is updated to avoid use of the term “must” for application requirements.

The DESCRIPTION is updated to explicitly state that *fseek()* sets the file-position indicator, and then on error the error indicator is set and *fseek()* fails. This is for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/42 is applied, updating the [EAGAIN] error in the ERRORS section from “the process would be delayed” to “the thread would be delayed”.

Issue 7

Changes are made related to support for finegrained timestamps.

fsetpos()

System Interfaces

31486 **NAME**31487 **fsetpos** — set current file position31488 **SYNOPSIS**31489 `#include <stdio.h>`31490 `int fsetpos(FILE *stream, const fpos_t *pos);`31491 **DESCRIPTION**

31492 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 31493 conflict between the requirements described here and the ISO C standard is unintentional. This
 31494 volume of POSIX.1-2008 defers to the ISO C standard.

31495 The *fsetpos()* function shall set the file position and state indicators for the stream pointed to by
 31496 *stream* according to the value of the object pointed to by *pos*, which the application shall ensure is
 31497 a value obtained from an earlier call to *fgetpos()* on the same stream. If a read or write error
 31498 occurs, the error indicator for the stream shall be set and *fsetpos()* fails.

31499 A successful call to the *fsetpos()* function shall clear the end-of-file indicator for the stream and
 31500 undo any effects of *ungetc()* on the same stream. After an *fsetpos()* call, the next operation on an
 31501 update stream may be either input or output.

31502 CX The behavior of *fsetpos()* on devices which are incapable of seeking is implementation-defined.
 31503 The value of the file offset associated with such a device is undefined.

31504 **RETURN VALUE**

31505 The *fsetpos()* function shall return 0 if it succeeds; otherwise, it shall return a non-zero value and
 31506 set *errno* to indicate the error.

31507 **ERRORS**

31508 CX The *fsetpos()* function shall fail if, either the *stream* is unbuffered or the *stream*'s buffer needed to
 31509 be flushed, and the call to *fsetpos()* causes an underlying *lseek()* or *write()* to be invoked, and:

31510 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the thread would be
 31511 delayed in the write operation.

31512 CX [EBADF] The file descriptor underlying the stream file is not open for writing or the
 31513 stream's buffer needed to be flushed and the file is not open.

31514 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

31515 XSI [EFBIG] An attempt was made to write a file that exceeds the file size limit of the
 31516 process.

31517 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 31518 offset maximum associated with the corresponding stream.

31519 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
 31520 was transferred.

31521 CX [EIO] A physical I/O error has occurred, or the process is a member of a background
 31522 process group attempting to perform a *write()* to its controlling terminal,
 31523 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the
 31524 process group of the process is orphaned. This error may also be returned
 31525 under implementation-defined conditions.

31526 CX [ENOSPC] There was no free space remaining on the device containing the file.

31527 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 31528 capabilities of the device.

31529 CX [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading
 31530 by any process; a SIGPIPE signal shall also be sent to the thread.

31531 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.

31532 EXAMPLES

31533 None.

31534 APPLICATION USAGE

31535 None.

31536 RATIONALE

31537 None.

31538 FUTURE DIRECTIONS

31539 None.

31540 SEE ALSO

31541 *fopen()*, *ftell()*, *lseek()*, *rewind()*, *ungetc()*, *write()*

31542 XBD <stdio.h>

31543 CHANGE HISTORY

31544 First released in Issue 4. Derived from the ISO C standard.

31545 Issue 6

31546 Extensions beyond the ISO C standard are marked.

31547 An additional [ESPIPE] error condition is added for sockets.

31548 The normative text is updated to avoid use of the term “must” for application requirements.

31549 The DESCRIPTION is updated to clarify that the error indicator is set for the stream on a read or
 31550 write error. This is for alignment with the ISO/IEC 9899:1999 standard.

31551 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/21 is applied, deleting an erroneous
 31552 [EINVAL] error case from the ERRORS section.

31553 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/43 is applied, updating the [EAGAIN]
 31554 error in the ERRORS section from “the process would be delayed” to “the thread would be
 31555 delayed”.

31556 Issue 7

31557 SD5-XSH-ERN-220 is applied.

fstat()*System Interfaces*31558 **NAME**31559 **fstat** — get file status31560 **SYNOPSIS**31561 `#include <sys/stat.h>`31562 `int fstat(int fildev, struct stat *buf);`31563 **DESCRIPTION**31564 The *fstat()* function shall obtain information about an open file associated with the file
31565 descriptor *fildev*, and shall write it to the area pointed to by *buf*.31566 SHM If *fildev* references a shared memory object, the implementation shall update in the **stat** structure
31567 pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the
31568 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be
31569 valid. The implementation may update other fields and flags.31570 TYM If *fildev* references a typed memory object, the implementation shall update in the **stat** structure
31571 pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the
31572 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be
31573 valid. The implementation may update other fields and flags.31574 The *buf* argument is a pointer to a **stat** structure, as defined in **<sys/stat.h>**, into which
31575 information is placed concerning the file.31576 For all other file types defined in this volume of POSIX.1-2008, the structure members *st_mode*,
31577 *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atim*, *st_ctim*, and *st_mtim* shall have meaningful values and the
31578 value of the *st_nlink* member shall be set to the number of links to the file.31579 An implementation that provides additional or alternative file access control mechanisms may,
31580 under implementation-defined conditions, cause *fstat()* to fail.31581 The *fstat()* function shall update any time-related fields (as described in XBD [Section 4.8](#), on
31582 page 109), before writing into the **stat** structure.31583 **RETURN VALUE**31584 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
31585 indicate the error.31586 **ERRORS**31587 The *fstat()* function shall fail if:31588 [EBADF] The *fildev* argument is not a valid file descriptor.

31589 [EIO] An I/O error occurred while reading from the file system.

31590 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file
31591 serial number cannot be represented correctly in the structure pointed to by
31592 *buf*.31593 The *fstat()* function may fail if:31594 [EOVERFLOW] One of the values is too large to store into the structure pointed to by the *buf*
31595 argument.

31596 **EXAMPLES**31597 **Obtaining File Status Information**

31598 The following example shows how to obtain file status information for a file named
 31599 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure. The
 31600 **/home/cnd/mod1** file is opened with read/write privileges and is passed to the open file
 31601 descriptor *filides*.

```
31602 #include <sys/types.h>
31603 #include <sys/stat.h>
31604 #include <fcntl.h>

31605 struct stat buffer;
31606 int      status;
31607 ...
31608 fildes = open("/home/cnd/mod1", O_RDWR);
31609 status = fstat(fildes, &buffer);
```

31610 **APPLICATION USAGE**

31611 None.

31612 **RATIONALE**

31613 None.

31614 **FUTURE DIRECTIONS**

31615 None.

31616 **SEE ALSO**

31617 *fstatat()*

31618 XBD Section 4.8 (on page 109), *<sys/stat.h>*, *<sys/types.h>*

31619 **CHANGE HISTORY**

31620 First released in Issue 1. Derived from Issue 1 of the SVID.

31621 **Issue 5**

31622 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

31623 Large File Summit extensions are added.

31624 **Issue 6**

31625 In the SYNOPSIS, the optional include of the *<sys/types.h>* header is removed.

31626 The following new requirements on POSIX implementations derive from alignment with the
 31627 Single UNIX Specification:

- 31628 • The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was
 31629 required for conforming implementations of previous POSIX specifications, it was not
 31630 required for UNIX applications.
- 31631 • The [EIO] mandatory error condition is added.
- 31632 • The [EOVERFLOW] mandatory error condition is added. This change is to support large
 31633 files.
- 31634 • The [EOVERFLOW] optional error condition is added.

31635 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
 31636 shared memory object semantics apply to typed memory objects.

fstat()*System Interfaces*31637 **Issue 7**

31638 XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st_nlink*
31639 applies.

31640 Changes are made related to support for finegrained timestamps.

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

System Interfaces

fstatat()

NAME

fstatat, lstat, stat — get file status

SYNOPSIS

```
#include <sys/stat.h>

int fstatat(int fd, const char *restrict path,
            struct stat *restrict buf, int flag);
int lstat(const char *restrict path, struct stat *restrict buf);
int stat(const char *restrict path, struct stat *restrict buf);
```

DESCRIPTION

The *stat()* function shall obtain information about the named file and write it to the area pointed to by the *buf* argument. The *path* argument points to a pathname naming a file. Read, write, or execute permission of the named file is not required. An implementation that provides additional or alternate file access control mechanisms may, under implementation-defined conditions, cause *stat()* to fail. In particular, the system may deny the existence of the file specified by *path*.

If the named file is a symbolic link, the *stat()* function shall continue pathname resolution using the contents of the symbolic link, and shall return information pertaining to the resulting file if the file exists.

The *buf* argument is a pointer to a **stat** structure, as defined in the **<sys/stat.h>** header, into which information is placed concerning the file.

The *stat()* function shall update any time-related fields (as described in XBD Section 4.8, on page 109), before writing into the **stat** structure.

SHM

If the named file is a shared memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be valid. The implementation may update other fields and flags.

TYM

If the named file is a typed memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be valid. The implementation may update other fields and flags.

For all other file types defined in this volume of POSIX.1-2008, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atim*, *st_ctim*, and *st_mtim* shall have meaningful values and the value of the member *st_nlink* shall be set to the number of links to the file.

The *lstat()* function shall be equivalent to *stat()*, except when *path* refers to a symbolic link. In that case *lstat()* shall return information about the link, while *stat()* shall return information about the file the link references.

For symbolic links, the *st_mode* member shall contain meaningful information when used with the file type macros. The file mode bits in *st_mode* are unspecified. The structure members *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atim*, *st_ctim*, and *st_mtim* shall have meaningful values and the value of the *st_nlink* member shall be set to the number of (hard) links to the symbolic link. The value of the *st_size* member shall be set to the length of the pathname contained in the symbolic link not including any terminating null byte.

The *fstatat()* function shall be equivalent to the *stat()* or *lstat()* function, depending on the value of *flag* (see below), except in the case where *path* specifies a relative path. In this case the status shall be retrieved from a file relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the

fstatat()*System Interfaces*

31687 function shall check whether directory searches are permitted using the current permissions of
 31688 the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH,
 31689 the function shall not perform the check.

31690 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
 31691 in **<fcntl.h>**:

31692 AT_SYMLINK_NOFOLLOW

31693 If *path* names a symbolic link, the status of the symbolic link is returned.

31694 If *fstatat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 31695 directory is used and the behavior shall be identical to a call to *stat()* or *lstat()* respectively,
 31696 depending on whether or not the AT_SYMLINK_NOFOLLOW bit is set in *flag*.

RETURN VALUE

31698 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 31699 return -1 and set *errno* to indicate the error.

ERRORS

31700 These functions shall fail if:

31702 [EACCES] Search permission is denied for a component of the path prefix.

31703 [EIO] An error occurred while reading from the file system.

31704 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 31705 argument.

31706 [ENAMETOOLONG]

31707 The length of a component of a pathname is longer than {NAME_MAX}.

31708 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

31709 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument
 31710 contains at least one non-*<slash>* character and ends with one or more trailing
 31711 *<slash>* characters and the last pathname component names an existing file
 31712 that is neither a directory nor a symbolic link to a directory.

31713 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file
 31714 serial number cannot be represented correctly in the structure pointed to by
 31715 *buf*.

31716 The *fstatat()* function shall fail if:

31717 [EACCES] *fd* was not opened with O_SEARCH and the permissions of the directory
 31718 underlying *fd* do not permit directory searches.

31719 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
 31720 neither AT_FDCWD nor a valid file descriptor open for reading or searching.

31721 These functions may fail if:

31722 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 31723 resolution of the *path* argument.

31724 [ENAMETOOLONG]

31725 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 31726 symbolic link produced an intermediate result with a length that exceeds
 31727 {PATH_MAX}.

- 31728 [EOVERFLOW] A value to be stored would overflow one of the members of the **stat** structure.
- 31729 The *fstatat()* function may fail if:
- 31730 [EINVAL] The value of the *flag* argument is not valid.
- 31731 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
- 31732 file descriptor associated with a directory.

31733 **EXAMPLES**31734 **Obtaining File Status Information**

31735 The following example shows how to obtain file status information for a file named

31736 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure.

```
31737 #include <sys/types.h>
31738 #include <sys/stat.h>
31739 #include <fcntl.h>
31740
31741 struct stat buffer;
31742 int      status;
31743 ...
31744 status = stat("/home/cnd/mod1", &buffer);
```

31744 **Getting Directory Information**

31745 The following example fragment gets status information for each entry in a directory. The call to

31746 the *stat()* function stores file information in the **stat** structure pointed to by *statbuf*. The lines

31747 that follow the *stat()* call format the fields in the **stat** structure for presentation to the user of the

31748 program.

```
31749 #include <sys/types.h>
31750 #include <sys/stat.h>
31751 #include <dirent.h>
31752 #include <pwd.h>
31753 #include <grp.h>
31754 #include <time.h>
31755 #include <locale.h>
31756 #include <langinfo.h>
31757 #include <stdio.h>
31758 #include <stdint.h>
31759
31760 struct dirent *dp;
31761 struct stat statbuf;
31762 struct passwd *pwd;
31763 struct group *grp;
31764 struct tm *tm;
31765 char datestring[256];
31766 ...
31767 /* Loop through directory entries. */
31768 while ((dp = readdir(dir)) != NULL) {
31769     /* Get entry's information. */
31770     if (stat(dp->d_name, &statbuf) == -1)
31771         continue;
```

fstatat()

System Interfaces

```

31771      /* Print out type, permissions, and number of links. */
31772      printf("%10.10s", sperm (statbuf.st_mode));
31773      printf("%4d", statbuf.st_nlink);
31774
31775      /* Print out owner's name if it is found using getpwuid(). */
31776      if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
31777          printf(" %-8.8s", pwd->pw_name);
31778      else
31779          printf(" %-8d", statbuf.st_uid);
31780
31781      /* Print out group name if it is found using getgrgid(). */
31782      if ((grp = getgrgid(statbuf.st_gid)) != NULL)
31783          printf(" %-8.8s", grp->gr_name);
31784      else
31785          printf(" %-8d", statbuf.st_gid);
31786
31787      /* Print size of file. */
31788      printf(" %9jd", (intmax_t)statbuf.st_size);
31789
31790      tm = localtime(&statbuf.st_mtime);
31791
31792      /* Get localized date string. */
31793      strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
31794
31795      printf(" %s %s\n", datestring, dp->d_name);
31796  }

```

Obtaining Symbolic Link Status Information

The following example shows how to obtain status information for a symbolic link named **/modules/pass1**. The structure variable *buffer* is defined for the **stat** structure. If the *path* argument specified the filename for the file pointed to by the symbolic link (**/home/cnd/mod1**), the results of calling the function would be the same as those returned by a call to the *stat()* function.

```

31797  #include <sys/stat.h>
31798
31799  struct stat buffer;
31800  int status;
31801  ...
31802  status = lstat("/modules/pass1", &buffer);

```

APPLICATION USAGE

None.

RATIONALE

The intent of the paragraph describing “additional or alternate file access control mechanisms” is to allow a secure implementation where a process with a label that does not dominate the file’s label cannot perform a *stat()* function. This is not related to read permission; a process with a label that dominates the file’s label does not need read permission. An implementation that supports write-up operations could fail *fstat()* function calls even though it has a valid file descriptor open for writing.

The *lstat()* function is not required to update the time-related fields if the named file is not a symbolic link. While the *st_uid*, *st_gid*, *st_atim*, *st_mtim*, and *st_ctim* members of the **stat** structure may apply to a symbolic link, they are not required to do so. No functions in POSIX.1-2008 are required to maintain any of these time fields.

The purpose of the *fstatat()* function is to obtain the status of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *stat()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *fstatat()* function it can be guaranteed that the file for which status is returned is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

access(), *chmod()*, *fdopendir()*, *fstat()*, *mknod()*, *readlink()*, *symlink()*

XBD Section 4.8 (on page 109), *<fcntl.h>*, *<sys/stat.h>*, *<sys/types.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

In the SYNOPSIS, the optional include of the *<sys/types.h>* header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EIO] mandatory error condition is added.
- The [ELOOP] mandatory error condition is added.
- The [EOVERFLOW] mandatory error condition is added. This change is to support large files.
- The [ENAMETOOLONG] and the second [EOVERFLOW] optional error conditions are added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Details are added regarding the treatment of symbolic links.
- The [ELOOP] optional error condition is added.

The normative text is updated to avoid use of the term “must” for application requirements.

The **restrict** keyword is added to the *stat()* prototype for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st_nlink* applies.

The *fstatat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

The *lstat()* function is now required to return meaningful data for symbolic links in all **stat**

fstatat()*System Interfaces*

- 31856 structure fields, except for the permission bits of *st_mode*.
- 31857 Changes are made to allow a directory to be opened for searching.
- 31858 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
- 31859 pathname exists but is not a directory or a symbolic link to a directory.

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

NAME

fstatvfs, statvfs — get file system information

SYNOPSIS

```
#include <sys/statvfs.h>
```

```
int fstatvfs(int fildev, struct statvfs *buf);
```

```
int statvfs(const char *restrict path, struct statvfs *restrict buf);
```

DESCRIPTION

The *fstatvfs()* function shall obtain information about the file system containing the file referenced by *fildev*.

The *statvfs()* function shall obtain information about the file system containing the file named by *path*.

For both functions, the *buf* argument is a pointer to a **statvfs** structure that shall be filled. Read, write, or execute permission of the named file is not required.

The following flags can be returned in the *f_flag* member:

ST_RDONLY Read-only file system.

ST_NOSUID Setuid/setgid bits ignored by *exec*.

It is unspecified whether all members of the **statvfs** structure have meaningful values on all file systems.

RETURN VALUE

Upon successful completion, *statvfs()* shall return 0. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *fstatvfs()* and *statvfs()* functions shall fail if:

[EIO] An I/O error occurred while reading the file system.

[EINTR] A signal was caught during execution of the function.

[EOVERFLOW] One of the values to be returned cannot be represented correctly in the structure pointed to by *buf*.

The *fstatvfs()* function shall fail if:

[EBADF] The *fildev* argument is not an open file descriptor.

The *statvfs()* function shall fail if:

[EACCES] Search permission is denied on a component of the path prefix.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]

The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix is not a directory, or the *path* argument contains at least one non-*<slash>* character and ends with one or more trailing *<slash>* characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

fstatvfs()*System Interfaces*

- 31900 The *statvfs()* function may fail if:
- 31901 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
31902 resolution of the *path* argument.
- 31903 [ENAMETOOLONG]
31904 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
31905 symbolic link produced an intermediate result with a length that exceeds
31906 {PATH_MAX}.

31907 **EXAMPLES**31908 **Obtaining File System Information Using fstatvfs()**

31909 The following example shows how to obtain file system information for the file system upon
31910 which the file named **/home/cnd/mod1** resides, using the *fstatvfs()* function. The
31911 **/home/cnd/mod1** file is opened with read/write privileges and the open file descriptor is passed
31912 to the *fstatvfs()* function.

```
31913 #include <sys/statvfs.h>
31914 #include <fcntl.h>

31915 struct statvfs buffer;
31916 int status;
31917 ...
31918 fildes = open("/home/cnd/mod1", O_RDWR);
31919 status = fstatvfs(fildes, &buffer);
```

31920 **Obtaining File System Information Using statvfs()**

31921 The following example shows how to obtain file system information for the file system upon
31922 which the file named **/home/cnd/mod1** resides, using the *statvfs()* function.

```
31923 #include <sys/statvfs.h>

31924 struct statvfs buffer;
31925 int status;
31926 ...
31927 status = statvfs("/home/cnd/mod1", &buffer);
```

31928 **APPLICATION USAGE**

31929 None.

31930 **RATIONALE**

31931 None.

31932 **FUTURE DIRECTIONS**

31933 None.

31934 **SEE ALSO**

31935 *chmod()*, *chown()*, *creat()*, *dup()*, *exec*, *fcntl()*, *link()*, *mknod()*, *open()*, *pipe()*, *read()*, *time()*,
31936 *unlink()*, *utime()*, *write()*

31937 XBD [<sys/statvfs.h>](#)

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Large File Summit extensions are added.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

The **restrict** keyword is added to the *statvfs()* prototype for alignment with the ISO/IEC 9899:1999 standard.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

SD5-XSH-ERN-68 is applied, correcting the EXAMPLES section.

The *fstatvfs()* and *statvfs()* functions are moved from the XSI option to the Base.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

fsync()*System Interfaces*31955 **NAME**31956 **fsync** — synchronize changes to a file31957 **SYNOPSIS**

```
31958 FSC      #include <unistd.h>
31959          int fsync(int fildes);
```

31960 **DESCRIPTION**

31961 The *fsync()* function shall request that all data for the open file descriptor named by *fildes* is to be
 31962 transferred to the storage device associated with the file described by *fildes*. The nature of the
 31963 transfer is implementation-defined. The *fsync()* function shall not return until the system has
 31964 completed that action or until an error is detected.

31965 SIO If `_POSIX_SYNCHRONIZED_IO` is defined, the *fsync()* function shall force all currently queued
 31966 I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O
 31967 completion state. All I/O operations shall be completed as defined for synchronized I/O file
 31968 integrity completion.

31969 **RETURN VALUE**

31970 Upon successful completion, *fsync()* shall return 0. Otherwise, -1 shall be returned and *errno* set
 31971 to indicate the error. If the *fsync()* function fails, outstanding I/O operations are not guaranteed
 31972 to have been completed.

31973 **ERRORS**31974 The *fsync()* function shall fail if:31975 [EBADF] The *fildes* argument is not a valid descriptor.31976 [EINTR] The *fsync()* function was interrupted by a signal.31977 [EINVAL] The *fildes* argument does not refer to a file on which this operation is possible.

31978 [EIO] An I/O error occurred while reading from or writing to the file system.

31979 In the event that any of the queued I/O operations fail, *fsync()* shall return the error conditions
 31980 defined for *read()* and *write()*.

31981 **EXAMPLES**

31982 None.

31983 **APPLICATION USAGE**

31984 The *fsync()* function should be used by programs which require modifications to a file to be
 31985 completed before continuing; for example, a program which contains a simple transaction
 31986 facility might use it to ensure that all modifications to a file or files caused by a transaction are
 31987 recorded.

31988 **RATIONALE**

31989 The *fsync()* function is intended to force a physical write of data from the buffer cache, and to
 31990 assure that after a system crash or other failure that all data up to the time of the *fsync()* call is
 31991 recorded on the disk. Since the concepts of “buffer cache”, “system crash”, “physical write”, and
 31992 “non-volatile storage” are not defined here, the wording has to be more abstract.

31993 If `_POSIX_SYNCHRONIZED_IO` is not defined, the wording relies heavily on the conformance
 31994 document to tell the user what can be expected from the system. It is explicitly intended that a
 31995 null implementation is permitted. This could be valid in the case where the system cannot assure
 31996 non-volatile storage under any circumstances or when the system is highly fault-tolerant and the
 31997 functionality is not required. In the middle ground between these extremes, *fsync()* might or
 31998 might not actually cause data to be written where it is safe from a power failure. The

conformance document should identify at least that one configuration exists (and how to obtain that configuration) where this can be assured for at least some files that the user can select to use for critical data. It is not intended that an exhaustive list is required, but rather sufficient information is provided so that if critical data needs to be saved, the user can determine how the system is to be configured to allow the data to be written to non-volatile storage.

It is reasonable to assert that the key aspects of *fsync()* are unreasonable to test in a test suite. That does not make the function any less valuable, just more difficult to test. A formal conformance test should probably force a system crash (power shutdown) during the test for this condition, but it needs to be done in such a way that automated testing does not require this to be done except when a formal record of the results is being made. It would also not be unreasonable to omit testing for *fsync()*, allowing it to be treated as a quality-of-implementation issue.

FUTURE DIRECTIONS

None.

SEE ALSO

sync()

XBD `<unistd.h>`

CHANGE HISTORY

First released in Issue 3.

Issue 5

Aligned with *fsync()* in the POSIX Realtime Extension. Specifically, the DESCRIPTION and RETURN VALUE sections are much expanded, and the ERRORS section is updated to indicate that *fsync()* can return the error conditions defined for *read()* and *write()*.

Issue 6

This function is marked as part of the File Synchronization option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] and [EIO] mandatory error conditions are added.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/44 is applied, applying an editorial rewording of the DESCRIPTION. No change in meaning is intended.

ftell()*System Interfaces*32029 **NAME**32030 **ftell, ftello** — return a file offset in a stream32031 **SYNOPSIS**32032 `#include <stdio.h>`32033 `long ftell(FILE *stream);`32034 CX `off_t ftello(FILE *stream);`32035 **DESCRIPTION**32036 CX The functionality described on this reference page is aligned with the ISO C standard. Any
32037 conflict between the requirements described here and the ISO C standard is unintentional. This
32038 volume of POSIX.1-2008 defers to the ISO C standard.32039 The *ftell()* function shall obtain the current value of the file-position indicator for the stream
32040 pointed to by *stream*.32041 CX The *ftello()* function shall be equivalent to *ftell()*, except that the return value is of type **off_t**.32042 **RETURN VALUE**32043 CX Upon successful completion, *ftell()* and *ftello()* shall return the current value of the file-position
32044 indicator for the stream measured in bytes from the beginning of the file.32045 CX Otherwise, *ftell()* and *ftello()* shall return **-1**, cast to **long** and **off_t** respectively, and set *errno* to
32046 indicate the error.32047 **ERRORS**32048 CX The *ftell()* and *ftello()* functions shall fail if:32049 CX **[EBADF]** The file descriptor underlying *stream* is not an open file descriptor.32050 CX **[EOVERFLOW]** For *ftell()*, the current file offset cannot be represented correctly in an object of
32051 type **long**.32052 CX **[EOVERFLOW]** For *ftello()*, the current file offset cannot be represented correctly in an object
32053 of type **off_t**.32054 CX **[ESPIPE]** The file descriptor underlying *stream* is associated with a pipe or FIFO.32055 The *ftell()* function may fail if:32056 CX **[ESPIPE]** The file descriptor underlying *stream* is associated with a socket.32057 **EXAMPLES**

32058 None.

32059 **APPLICATION USAGE**

32060 None.

32061 **RATIONALE**

32062 None.

32063 **FUTURE DIRECTIONS**

32064 None.

32065 **SEE ALSO**32066 *fgetpos()*, *fopen()*, *fseek()*, *lseek()*32067 XBD `<stdio.h>`

32068 **CHANGE HISTORY**

32069 First released in Issue 1. Derived from Issue 1 of the SVID.

32070 **Issue 5**

32071 Large File Summit extensions are added.

32072 **Issue 6**

32073 Extensions beyond the ISO C standard are marked.

32074 The following new requirements on POSIX implementations derive from alignment with the
32075 Single UNIX Specification:

- 32076 • The *ftello()* function is added.
- 32077 • The [Eoverflow] error conditions are added.

32078 An additional [ESPIPE] error condition is added for sockets.

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

ftok()*System Interfaces*32079 **NAME**32080 **ftok** — generate an IPC key32081 **SYNOPSIS**

```
32082 xSI      #include <sys/ipc.h>
32083          key_t ftok(const char *path, int id);
```

32084 **DESCRIPTION**

32085 The *ftok()* function shall return a key based on *path* and *id* that is usable in subsequent calls to
 32086 *msgget()*, *semget()*, and *shmget()*. The application shall ensure that the *path* argument is the
 32087 pathname of an existing file that the process is able to *stat()*.

32088 The *ftok()* function shall return the same key value for all paths that name the same file, when
 32089 called with the same *id* value, and return different key values when called with different *id*
 32090 values or with paths that name different files existing on the same file system at the same time. It
 32091 is unspecified whether *ftok()* shall return the same key value when called again after the file
 32092 named by *path* is removed and recreated with the same name.

32093 Only the low-order 8-bits of *id* are significant. The behavior of *ftok()* is unspecified if these bits
 32094 are 0.

32095 **RETURN VALUE**

32096 Upon successful completion, *ftok()* shall return a key. Otherwise, *ftok()* shall return (**key_t**)-1
 32097 and set *errno* to indicate the error.

32098 **ERRORS**

32099 The *ftok()* function shall fail if:

- 32100 [EACCES] Search permission is denied for a component of the path prefix.
- 32101 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 32102 argument.
- 32103 [ENAMETOOLONG]
 32104 The length of a component of a pathname is longer than {NAME_MAX}.
- 32105 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 32106 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument
 32107 contains at least one non-*<slash>* character and ends with one or more trailing
 32108 *<slash>* characters and the last pathname component names an existing file
 32109 that is neither a directory nor a symbolic link to a directory.

32110 The *ftok()* function may fail if:

- 32111 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 32112 resolution of the *path* argument.
- 32113 [ENAMETOOLONG]
 32114 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 32115 symbolic link produced an intermediate result with a length that exceeds
 32116 {PATH_MAX}.

32117 **EXAMPLES**32118 **Getting an IPC Key**

32119 The following example gets a unique key that can be used by the IPC functions *semget()*,
 32120 *msgget()*, and *shmget()*. The key returned by *ftok()* for this example is based on the ID value *S*
 32121 and the pathname */tmp*.

```
32122 #include <sys/ipc.h>
32123 ...
32124 key_t key;
32125 char *path = "/tmp";
32126 int id = 'S';
32127 key = ftok(path, id);
```

32128 **Saving an IPC Key**

32129 The following example gets a unique key based on the pathname */tmp* and the ID value *a*. It
 32130 also assigns the value of the resulting key to the *semkey* variable so that it will be available to a
 32131 later call to *semget()*, *msgget()*, or *shmget()*.

```
32132 #include <sys/ipc.h>
32133 ...
32134 key_t semkey;
32135 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
32136     perror("IPC error: ftok"); exit(1);
32137 }
```

32138 **APPLICATION USAGE**

32139 For maximum portability, *id* should be a single-byte character.

32140 **RATIONALE**

32141 None.

32142 **FUTURE DIRECTIONS**

32143 None.

32144 **SEE ALSO**

32145 *msgget()*, *semget()*, *shmget()*

32146 XBD *<sys/ipc.h>*

32147 **CHANGE HISTORY**

32148 First released in Issue 4, Version 2.

32149 **Issue 5**

32150 Moved from X/OPEN UNIX extension to BASE.

32151 **Issue 6**

32152 The normative text is updated to avoid use of the term “must” for application requirements.

32153 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
 32154 [ELOOP] error condition is added.

ftok()*System Interfaces***Issue 7**

32155 Austin Group Interpretation 1003.1-2001 #143 is applied.

32156
32157 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
32158 pathname exists but is not a directory or a symbolic link to a directory.

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

NAME

ftruncate — truncate a file to a specified length

SYNOPSIS

```
#include <unistd.h>

int ftruncate(int fildevs, off_t length);
```

DESCRIPTION

If *fildevs* is not a valid file descriptor open for writing, the *ftruncate()* function shall fail.

If *fildevs* refers to a regular file, the *ftruncate()* function shall cause the size of the file to be truncated to *length*. If the size of the file previously exceeded *length*, the extra data shall no longer be available to reads on the file. If the file previously was smaller than this size, *ftruncate()* shall increase the size of the file. If the file size is increased, the extended area shall appear as if it were zero-filled. The value of the seek pointer shall not be modified by a call to *ftruncate()*.

Upon successful completion, if *fildevs* refers to a regular file, *ftruncate()* shall mark for update the last data modification and last file status change timestamps of the file and the S_ISUID and S_ISGID bits of the file mode may be cleared. If the *ftruncate()* function is unsuccessful, the file is unaffected.

XSI If the request would cause the file size to exceed the soft file size limit for the process, the request shall fail and the implementation shall generate the SIGXFSZ signal for the thread.

If *fildevs* refers to a directory, *ftruncate()* shall fail.

If *fildevs* refers to any other file type, except a shared memory object, the result is unspecified.

SHM If *fildevs* refers to a shared memory object, *ftruncate()* shall set the size of the shared memory object to *length*.

SHM If the effect of *ftruncate()* is to decrease the size of a memory mapped file or a shared memory object and whole pages beyond the new end were previously mapped, then the whole pages beyond the new end shall be discarded.

References to discarded pages shall result in the generation of a SIGBUS signal.

If the effect of *ftruncate()* is to increase the size of a memory object, it is unspecified whether the contents of any mapped pages between the old end-of-file and the new are flushed to the underlying object.

RETURN VALUE

Upon successful completion, *ftruncate()* shall return 0; otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *ftruncate()* function shall fail if:

[EINTR] A signal was caught during execution.

[EINVAL] The *length* argument was less than 0.

[EFBIG] or [EINVAL]

The *length* argument was greater than the maximum file size.

[EFBIG] The file is a regular file and *length* is greater than the offset maximum established in the open file description associated with *fildevs*.

ftruncate()*System Interfaces*

32200 [EIO] An I/O error occurred while reading from or writing to a file system.

32201 [EBADF] or [EINVAL]

32202 The *files* argument is not a file descriptor open for writing.

32203 **EXAMPLES**

32204 None.

32205 **APPLICATION USAGE**

32206 None.

32207 **RATIONALE**

32208 None.

32209 **FUTURE DIRECTIONS**

32210 None.

32211 **SEE ALSO**

32212 *open()*, *truncate()*

32213 XBD <unistd.h>

32214 **CHANGE HISTORY**

32215 First released in Issue 4, Version 2.

32216 **Issue 5**

32217 Moved from X/OPEN UNIX extension to BASE and aligned with *ftruncate()* in the POSIX
32218 Realtime Extension. Specifically, the DESCRIPTION is extensively reworded and [EROFS] is
32219 added to the list of mandatory errors that can be returned by *ftruncate()*.

32220 Large File Summit extensions are added.

32221 **Issue 6**

32222 The *truncate()* function is split out into a separate reference page.

32223 The following new requirements on POSIX implementations derive from alignment with the
32224 Single UNIX Specification:

- 32225 • The DESCRIPTION is changed to indicate that if the file size is changed, and if the file is a
32226 regular file, the S_ISUID and S_ISGID bits in the file mode may be cleared.

32227 The following changes were made to align with the IEEE P1003.1a draft standard:

- 32228 • The DESCRIPTION text is updated.

32229 XSI-conformant systems are required to increase the size of the file if the file was previously
32230 smaller than the size requested.

32231 **Issue 7**

32232 Austin Group Interpretation 1003.1-2001 #056 is applied, revising the ERRORS section (although
32233 the [EINVAL] “may fail” error was subsequently removed during review of the XSI option).

32234 Functionality relating to the Memory Protection and Memory Mapped Files options is moved to
32235 the Base.

32236 The DESCRIPTION is updated so that a call to *ftruncate()* when the file is smaller than the size
32237 requested will increase the size of the file. Previously, non-XSI-conforming implementations
32238 were allowed to increase the size of the file or fail.

32239 Changes are made related to support for finegrained timestamps.

*System Interfaces***ftrylockfile()**32240 **NAME**

32241 ftrylockfile — stdio locking functions

32242 **SYNOPSIS**

```
32243 CX #include <stdio.h>  
32244 int ftrylockfile(FILE *file);
```

32245 **DESCRIPTION**32246 Refer to *flockfile()*.

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

ftw()*System Interfaces*32247 **NAME**32248 **ftw** — traverse (walk) a file tree32249 **SYNOPSIS**

```

32250 OB XSI #include <ftw.h>
32251
32251 int ftw(const char *path, int (*fn)(const char *,
32252      const struct stat *ptr, int flag), int ndirs);

```

32253 **DESCRIPTION**

32254 The *ftw()* function shall recursively descend the directory hierarchy rooted in *path*. For each
 32255 object in the hierarchy, *ftw()* shall call the function pointed to by *fn*, passing it a pointer to a null-
 32256 terminated character string containing the name of the object, a pointer to a **stat** structure
 32257 containing information about the object, filled in as if *stat()* or *lstat()* had been called to retrieve
 32258 the information. Possible values of the integer, defined in the **<ftw.h>** header, are:

32259 **FTW_D** For a directory.32260 **FTW_DNR** For a directory that cannot be read.32261 **FTW_F** For a file.32262 **FTW_SL** For a symbolic link (but see also **FTW_NS** below).

32263 **FTW_NS** For an object other than a symbolic link on which *stat()* could not successfully be
 32264 executed. If the object is a symbolic link and *stat()* failed, it is unspecified whether
 32265 *ftw()* passes **FTW_SL** or **FTW_NS** to the user-supplied function.

32266 If the integer is **FTW_DNR**, descendants of that directory shall not be processed. If the integer is
 32267 **FTW_NS**, the **stat** structure contains undefined values. An example of an object that would
 32268 cause **FTW_NS** to be passed to the function pointed to by *fn* would be a file in a directory with
 32269 read but without execute (search) permission.

32270 The *ftw()* function shall visit a directory before visiting any of its descendants.32271 The *ftw()* function shall use at most one file descriptor for each level in the tree.32272 The argument *ndirs* should be in the range [1,{**OPEN_MAX**}].

32273 The tree traversal shall continue until either the tree is exhausted, an invocation of *fn* returns a
 32274 non-zero value, or some error, other than [**EACCES**], is detected within *ftw()*.

32275 The *ndirs* argument shall specify the maximum number of directory streams or file descriptors
 32276 or both available for use by *ftw()* while traversing the tree. When *ftw()* returns it shall close any
 32277 directory streams and file descriptors it uses not counting any opened by the application-
 32278 supplied *fn* function.

32279 The results are unspecified if the application-supplied *fn* function does not preserve the current
 32280 working directory.

32281 The *ftw()* function need not be thread-safe.32282 **RETURN VALUE**

32283 If the tree is exhausted, *ftw()* shall return 0. If the function pointed to by *fn* returns a non-zero
 32284 value, *ftw()* shall stop its tree traversal and return whatever value was returned by the function
 32285 pointed to by *fn*. If *ftw()* detects an error, it shall return -1 and set *errno* to indicate the error.

32286 If *ftw()* encounters an error other than [**EACCES**] (see **FTW_DNR** and **FTW_NS** above), it shall
 32287 return -1 and set *errno* to indicate the error. The external variable *errno* may contain any error
 32288 value that is possible when a directory is opened or when one of the *stat* functions is executed on

32289 a directory or file.

32290 ERRORS

32291 The *ftw()* function shall fail if:

32292 [EACCES] Search permission is denied for any component of *path* or read permission is
32293 denied for *path*.

32294 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
32295 argument.

32296 [ENAMETOOLONG]

32297 The length of a component of a pathname is longer than {NAME_MAX}.

32298 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

32299 [ENOTDIR] A component of *path* is not a directory.

32300 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current
32301 programming environment for one or more files found in the file hierarchy.

32302 The *ftw()* function may fail if:

32303 [EINVAL] The value of the *ndirs* argument is invalid.

32304 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
32305 resolution of the *path* argument.

32306 [ENAMETOOLONG]

32307 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
32308 symbolic link produced an intermediate result with a length that exceeds
32309 {PATH_MAX}.

32310 In addition, if the function pointed to by *fn* encounters system errors, *errno* may be set
32311 accordingly.

32312 EXAMPLES

32313 Walking a Directory Structure

32314 The following example walks the current directory structure, calling the *fn* function for every
32315 directory entry, using at most 10 file descriptors:

```
32316 #include <ftw.h>
32317 ...
32318 if (ftw(".", fn, 10) != 0) {
32319     perror("ftw"); exit(2);
32320 }
```

32321 APPLICATION USAGE

32322 The *ftw()* function may allocate dynamic storage during its operation. If *ftw()* is forcibly
32323 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *fn*
32324 or an interrupt routine, *ftw()* does not have a chance to free that storage, so it remains
32325 permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has
32326 occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next
32327 invocation.

32328 Applications should use the *nftw()* function instead of the obsolescent *ftw()* function.

ftw()*System Interfaces*32329 **RATIONALE**

32330 None.

32331 **FUTURE DIRECTIONS**32332 The *ftw()* function may be removed in a future version.32333 **SEE ALSO**32334 *fdopendir()*, *fstatat()*, *longjmp()*, *nftw()*, *siglongjmp()*32335 XBD *<ftw.h>*, *<sys/stat.h>*32336 **CHANGE HISTORY**

32337 First released in Issue 1. Derived from Issue 1 of the SVID.

32338 **Issue 5**

32339 UX codings in the DESCRIPTION, RETURN VALUE, and ERRORS sections are changed to EX.

32340 **Issue 6**

32341 The ERRORS section is updated as follows:

- 32342 • The wording of the mandatory [ELOOP] error condition is updated.
- 32343 • A second optional [ELOOP] error condition is added.
- 32344 • The [Eoverflow] mandatory error condition is added.

32345 A note is added to the DESCRIPTION indicating that this function need not be reentrant, and
 32346 that the results are unspecified if the application-supplied *fn* function does not preserve the
 32347 current working directory.

32348 **Issue 7**

32349 Austin Group Interpretations 1003.1-2001 #143 and #156 are applied.

32350 SD5-XBD-ERN-61 is applied.

32351 The *ftw()* function is marked obsolescent.

*System Interfaces***funlockfile()**32352 **NAME**

32353 funlockfile — stdio locking functions

32354 **SYNOPSIS**

```
32355 CX #include <stdio.h>  
32356 void funlockfile(FILE *file);
```

32357 **DESCRIPTION**32358 Refer to *flockfile()*.

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

futimens()*System Interfaces*32359 **NAME**

32360 futimens, utimensat, utimes — set file access and modification times

32361 **SYNOPSIS**

```

32362 #include <sys/stat.h>
32363 int futimens(int fd, const struct timespec times[2]);
32364 int utimensat(int fd, const char *path, const struct timespec times[2],
32365             int flag);
32366 XSI #include <sys/time.h>
32367 int utimes(const char *path, const struct timeval times[2]);

```

32368 **DESCRIPTION**

32369 The *futimens()* and *utimensat()* functions shall set the access and modification times of a file to
 32370 the values of the *times* argument. The *futimens()* function changes the times of the file associated
 32371 with the file descriptor *fd*. The *utimensat()* function changes the times of the file pointed to by
 32372 the *path* argument, relative to the directory associated with the file descriptor *fd*. Both functions
 32373 allow time specifications accurate to the nanosecond.

32374 For *futimens()* and *utimensat()*, the *times* argument is an array of two **timespec** structures. The
 32375 first array member represents the date and time of last access, and the second member
 32376 represents the date and time of last modification. The times in the **timespec** structure are
 32377 measured in seconds and nanoseconds since the Epoch. The file's relevant timestamp shall be set
 32378 to the greatest value supported by the file system that is not greater than the specified time.

32379 If the *tv_nsec* field of a **timespec** structure has the special value **UTIME_NOW**, the file's relevant
 32380 timestamp shall be set to the greatest value supported by the file system that is not greater than
 32381 the current time. If the *tv_nsec* field has the special value **UTIME_OMIT**, the file's relevant
 32382 timestamp shall not be changed. In either case, the *tv_sec* field shall be ignored.

32383 If the *times* argument is a null pointer, both the access and modification timestamps shall be set
 32384 to the greatest value supported by the file system that is not greater than the current time. If
 32385 *utimensat()* is passed a relative path in the *path* argument, the file to be used shall be relative to
 32386 the directory associated with the file descriptor *fd* instead of the current working directory. If the
 32387 file descriptor was opened without **O_SEARCH**, the function shall check whether directory
 32388 searches are permitted using the current permissions of the directory underlying the file
 32389 descriptor. If the file descriptor was opened with **O_SEARCH**, the function shall not perform the
 32390 check.

32391 If *utimensat()* is passed the special value **AT_FDCWD** in the *fd* parameter, the current working
 32392 directory shall be used.

32393 Only a process with the effective user ID equal to the user ID of the file, or with write access to
 32394 the file, or with appropriate privileges may use *futimens()* or *utimensat()* with a null pointer as
 32395 the *times* argument or with both *tv_nsec* fields set to the special value **UTIME_NOW**. Only a
 32396 process with the effective user ID equal to the user ID of the file or with appropriate privileges
 32397 may use *futimens()* or *utimensat()* with a non-null *times* argument that does not have both
 32398 *tv_nsec* fields set to **UTIME_NOW** and does not have both *tv_nsec* fields set to **UTIME_OMIT**. If
 32399 both *tv_nsec* fields are set to **UTIME_OMIT**, no ownership or permissions check shall be
 32400 performed for the file, but other error conditions may still be detected (including **[EACCES]**
 32401 errors related to the path prefix).

32402 Values for the *flag* argument of *utimensat()* are constructed by a bitwise-inclusive OR of flags
 32403 from the following list, defined in **<fcntl.h>**:

System Interfaces

futimens()

- 32404 AT_SYMLINK_NOFOLLOW
 32405 If *path* names a symbolic link, then the access and modification times of the symbolic link
 32406 are changed.
- 32407 Upon completion, *futimens()* and *utimensat()* shall mark the last file status change timestamp for
 32408 update.
- 32409 The *utimes()* function shall be equivalent to the *utimensat()* function with the special value
 32410 AT_FDCWD as the *fd* argument and the *flag* argument set to zero, except that the *times* argument
 32411 is a **timeval** structure rather than a **timespec** structure, and accuracy is only to the microsecond,
 32412 not nanosecond, and rounding towards the nearest second may occur.
- 32413 **RETURN VALUE**
 32414 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 32415 return -1 and set *errno* to indicate the error. If -1 is returned, the file times shall not be affected.
- 32416 **ERRORS**
 32417 These functions shall fail if:
- 32418 [EACCES] The *times* argument is a null pointer, or both *tv_nsec* values are UTIME_NOW,
 32419 and the effective user ID of the process does not match the owner of the file
 32420 and write access is denied.
- 32421 [EINVAL] Either of the *times* argument structures specified a *tv_nsec* value that was
 32422 neither UTIME_NOW nor UTIME_OMIT, and was a value less than zero or
 32423 greater than or equal to 1 000 million.
- 32424 [EINVAL] A new file timestamp would be a value whose *tv_sec* component is not a value
 32425 supported by the file system.
- 32426 [EPERM] The *times* argument is not a null pointer, does not have both *tv_nsec* fields set
 32427 to UTIME_NOW, does not have both *tv_nsec* fields set to UTIME_OMIT, the
 32428 calling process' effective user ID has write access to the file but does not match
 32429 the owner of the file, and the calling process does not have appropriate
 32430 privileges.
- 32431 [EROFS] The file system containing the file is read-only.
- 32432 The *futimens()* function shall fail if:
- 32433 [EBADF] The *fd* argument is not a valid file descriptor.
- 32434 The *utimensat()* function shall fail if:
- 32435 [EACCES] *fd* was not opened with O_SEARCH and the permissions of the directory
 32436 underlying *fd* do not permit directory searches.
- 32437 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
 32438 neither AT_FDCWD nor a valid file descriptor open for reading or searching.
- 32439 The *utimensat()* and *utimes()* functions shall fail if:
- 32440 [EACCES] Search permission is denied by a component of the path prefix.
- 32441 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 32442 argument.
- 32443 [ENAMETOOLONG]
 32444 The length of a component of a pathname is longer than {NAME_MAX}.

futimens()*System Interfaces*

- 32445 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 32446 [ENOTDIR] A component of the *path* prefix is not a directory, or the *path* argument
32447 contains at least one non-`<slash>` character and ends with one or more trailing
32448 `<slash>` characters and the last pathname component names an existing file
32449 that is neither a directory nor a symbolic link to a directory.
- 32450 The *utimensat()* and *utimes()* functions may fail if:
- 32451 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
32452 resolution of the *path* argument.
- 32453 [ENAMETOOLONG]
32454 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
32455 symbolic link produced an intermediate result with a length that exceeds
32456 {PATH_MAX}.
- 32457 The *utimensat()* function may fail if:
- 32458 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
32459 file descriptor associated with a directory.

EXAMPLES

32460 None.

APPLICATION USAGE

32463 None.

RATIONALE

32465 The purpose of the *utimensat()* function is to set the access and modification time of files in
32466 directories other than the current working directory without exposure to race conditions. Any
32467 part of the path of a file could be changed in parallel to a call to *utimes()*, resulting in unspecified
32468 behavior. By opening a file descriptor for the target directory and using the *utimensat()* function
32469 it can be guaranteed that the changed file is located relative to the desired directory.

32470 The standard developers considered including a special case for the permissions required by
32471 *utimensat()* when one *tv_nsec* field is `UTIME_NOW` and the other is `UTIME_OMIT`. One
32472 possibility would be to include this case in with the cases where *times* is a null pointer or both
32473 fields are `UTIME_NOW`, where the call is allowed if the process has write permission for the file.
32474 However, associating write permission with an update to just the last data access timestamp
32475 (which is normally updated by *read()*) did not seem appropriate. The other possibility would be
32476 to specify that this one case is allowed if the process has read permission, but this was felt to be
32477 too great a departure from the *utime()* and *utimes()* functions on which *utimensat()* is based. If
32478 an application needs to set the last data access timestamp to the current time for a file on which
32479 it has read permission but is not the owner, it can do so by opening the file, reading one or more
32480 bytes (or reading a directory entry, if the file is a directory), and then closing it.

FUTURE DIRECTIONS

32482 None.

SEE ALSO

32484 *read()*, *utime()*

32485 XBD `<fcntl.h>`, `<sys/stat.h>`, `<sys/time.h>`

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

This function is marked LEGACY.

The normative text is updated to avoid use of the term “must” for application requirements.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The LEGACY marking is removed.

The *utimensat()* function (renamed from *futimesat()*) is added from The Open Group Technical Standard, 2006, Extended API Set Part 2, and changed to allow modifying a symbolic link by adding a *flag* argument.

The *futimens()* function is added.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

fwide()*System Interfaces*32506 **NAME**32507 **fwide** — set stream orientation32508 **SYNOPSIS**32509 `#include <stdio.h>`32510 `#include <wchar.h>`32511 `int fwide(FILE *stream, int mode);`32512 **DESCRIPTION**

32513 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 32514 conflict between the requirements described here and the ISO C standard is unintentional. This
 32515 volume of POSIX.1-2008 defers to the ISO C standard.

32516 The *fwide()* function shall determine the orientation of the stream pointed to by *stream*. If *mode* is
 32517 greater than zero, the function first attempts to make the stream wide-oriented. If *mode* is less
 32518 than zero, the function first attempts to make the stream byte-oriented. Otherwise, *mode* is zero
 32519 and the function does not alter the orientation of the stream.

32520 If the orientation of the stream has already been determined, *fwide()* shall not change it.

32521 CX Since no return value is reserved to indicate an error, an application wishing to check for error
 32522 situations should set *errno* to 0, then call *fwide()*, then check *errno*, and if it is non-zero, assume
 32523 an error has occurred.

32524 **RETURN VALUE**

32525 The *fwide()* function shall return a value greater than zero if, after the call, the stream has wide-
 32526 orientation, a value less than zero if the stream has byte-orientation, or zero if the stream has no
 32527 orientation.

32528 **ERRORS**

32529 The *fwide()* function may fail if:

32530 CX [EBADF] The *stream* argument is not a valid stream.

32531 **EXAMPLES**

32532 None.

32533 **APPLICATION USAGE**

32534 A call to *fwide()* with *mode* set to zero can be used to determine the current orientation of a
 32535 stream.

32536 **RATIONALE**

32537 None.

32538 **FUTURE DIRECTIONS**

32539 None.

32540 **SEE ALSO**

32541 XBD `<stdio.h>`, `<wchar.h>`

32542 **CHANGE HISTORY**

32543 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 32544 (E).

32545 **Issue 6**

32546 Extensions beyond the ISO C standard are marked.

NAME

fwprintf, swprintf, wprintf — print formatted wide-character output

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int fwprintf(FILE *restrict stream, const wchar_t *restrict format, ...);
int swprintf(wchar_t *restrict ws, size_t n,
             const wchar_t *restrict format, ...);
int wprintf(const wchar_t *restrict format, ...);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *fwprintf()* function shall place output on the named output *stream*. The *wprintf()* function shall place output on the standard output stream *stdout*. The *swprintf()* function shall place output followed by the null wide character in consecutive wide characters starting at **ws*; no more than *n* wide characters shall be written, including a terminating null wide character, which is always added (unless *n* is zero).

Each of these functions shall convert, format, and print its arguments under control of the *format* wide-character string. The *format* is composed of zero or more directives: *ordinary wide-characters*, which are simply copied to the output stream, and *conversion specifications*, each of which results in the fetching of zero or more arguments. The results are undefined if there are insufficient arguments for the *format*. If the *format* is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier wide character % (see below) is replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}], giving the position of the argument in the argument list. This feature provides for the definition of *format* wide-character strings that select arguments in an order appropriate to specific languages (see the EXAMPLES section).

The *format* can contain either numbered argument specifications (that is, "%n\$" and "%m\$"), or unnumbered argument conversion specifications (that is, % and *), but not both. The only exception to this is that %% can be mixed with the "%n\$" form. The results of mixing numbered and unnumbered argument specifications in a *format* wide-character string are undefined. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*−1)th, are specified in the *format* wide-character string.

In *format* wide-character strings containing the "%n\$" form of conversion specification, numbered arguments in the argument list can be referenced from the *format* wide-character string as many times as required.

In *format* wide-character strings containing the % form of conversion specification, each argument in the argument list shall be used exactly once.

All forms of the *fwprintf()* function allow for the insertion of a locale-dependent radix character in the output string, output as a wide-character value. The radix character is defined in the locale of the process (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> ('.').

fwprintf()*System Interfaces*

32592	CX	Each conversion specification is introduced by the ' % ' wide character or by the wide-character sequence "%n\$", after which the following appear in sequence:
32593		
32594		• Zero or more <i>flags</i> (in any order), which modify the meaning of the conversion specification.
32595		
32596		• An optional minimum <i>field width</i> . If the converted value has fewer wide characters than the field width, it shall be padded with <space> characters by default on the left; it shall be padded on the right, if the left-adjustment flag (' - '), described below, is given to the field width. The field width takes the form of an <asterisk> (' * '), described below, or a decimal integer.
32597		
32598		
32599		
32600		
32601		• An optional <i>precision</i> that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversion specifiers; the number of digits to appear after the radix character for the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for the g and G conversion specifiers; or the maximum number of wide characters to be printed from a string in the s conversion specifiers. The precision takes the form of a <period> (' . ') followed either by an <asterisk> (' * '), described below, or an optional decimal digit string, where a null digit string is treated as 0. If a precision appears with any other conversion wide character, the behavior is undefined.
32602		
32603		
32604		
32605		
32606		
32607		
32608		
32609		• An optional length modifier that specifies the size of the argument.
32610		• A <i>conversion specifier</i> wide character that indicates the type of conversion to be applied.
32611		A field width, or precision, or both, may be indicated by an <asterisk> (' * '). In this case an argument of type int supplies the field width or precision. Applications shall ensure that arguments specifying field width, or precision, or both appear in that order before the argument, if any, to be converted. A negative field width is taken as a ' - ' flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In <i>format</i> wide-character strings containing the "%n\$" form of a conversion specification, a field width or precision may be indicated by the sequence "%m\$", where <i>m</i> is a decimal integer in the range [1,{NL_ARGMAX}] giving the position in the argument list (after the <i>format</i> argument) of an integer argument containing the field width or precision, for example:
32612		
32613		
32614		
32615	CX	
32616		
32617		
32618		
32619		
32620		<pre>wprintf(L"%1\$d:%2\$. *3\$d:%4\$. *3\$d\n", hour, min, precision, sec);</pre>
32621		The flag wide characters and their meanings are:
32622	CX	' (The <apostrophe>.) The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping wide characters. For other conversions, the behavior is undefined. The numeric grouping wide character is used.
32623		
32624		
32625		
32626		- The result of the conversion shall be left-justified within the field. The conversion shall be right-justified if this flag is not specified.
32627		
32628		+ The result of a signed conversion shall always begin with a sign (' + ' or ' - '). The conversion shall begin with a sign only when a negative value is converted if this flag is not specified.
32629		
32630		
32631		<space> If the first wide character of a signed conversion is not a sign, or if a signed conversion results in no wide characters, a <space> shall be prefixed to the result. This means that if the <space> and ' + ' flags both appear, the <space> flag shall be ignored.
32632		
32633		
32634		# Specifies that the value is to be converted to an alternative form. For o conversion, it increases the precision (if necessary) to force the first digit of the result to be 0. For x or X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e,
32635		
32636		

32637		E, f, F, g, and G conversion specifiers, the result shall always contain a radix character, even if no digits follow it. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For g and G conversion specifiers, trailing zeros shall <i>not</i> be removed from the result as they normally are. For other conversion specifiers, the behavior is undefined.
32638		
32639		
32640		
32641		
32642	0	For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the '0' and '-' flags both appear, the '0' flag shall be ignored. For d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag shall be ignored. If the '0' and <apostrophe> flags both appear, the grouping wide characters are inserted before zero padding. For other conversions, the behavior is undefined.
32643		
32644		
32645		
32646	CX	
32647		
32648		
32649		The length modifiers and their meanings are:
32650	hh	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a signed char or unsigned char argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to signed char or unsigned char before printing); or that a following n conversion specifier applies to a pointer to a signed char argument.
32651		
32652		
32653		
32654		
32655	h	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a short or unsigned short argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to short or unsigned short before printing); or that a following n conversion specifier applies to a pointer to a short argument.
32656		
32657		
32658		
32659		
32660	l (ell)	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long or unsigned long argument; that a following n conversion specifier applies to a pointer to a long argument; that a following c conversion specifier applies to a wint_t argument; that a following s conversion specifier applies to a pointer to a wchar_t argument; or has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.
32661		
32662		
32663		
32664		
32665	ll (ell-ell)	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long long or unsigned long long argument; or that a following n conversion specifier applies to a pointer to a long long argument.
32666		
32667		
32668		
32669	j	Specifies that a following d, i, o, u, x, or X conversion specifier applies to an intmax_t or uintmax_t argument; or that a following n conversion specifier applies to a pointer to an intmax_t argument.
32670		
32671		
32672	z	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a size_t or the corresponding signed integer type argument; or that a following n conversion specifier applies to a pointer to a signed integer type corresponding to a size_t argument.
32673		
32674		
32675	t	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a ptrdiff_t or the corresponding unsigned type argument; or that a following n conversion specifier applies to a pointer to a ptrdiff_t argument.
32676		
32677		
32678	L	Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a long double argument.
32679		

fwprintf()*System Interfaces*

32680		If a length modifier appears with any conversion specifier other than as specified above, the
32681		behavior is undefined.
32682		The conversion specifiers and their meanings are:
32683	d, i	The int argument shall be converted to a signed decimal in the style "[−] dddd". The
32684		precision specifies the minimum number of digits to appear; if the value being
32685		converted can be represented in fewer digits, it shall be expanded with leading zeros.
32686		The default precision shall be 1. The result of converting zero with an explicit precision
32687		of zero shall be no wide characters.
32688	o	The unsigned argument shall be converted to unsigned octal format in the style
32689		"ddd". The precision specifies the minimum number of digits to appear; if the value
32690		being converted can be represented in fewer digits, it shall be expanded with leading
32691		zeros. The default precision shall be 1. The result of converting zero with an explicit
32692		precision of zero shall be no wide characters.
32693	u	The unsigned argument shall be converted to unsigned decimal format in the style
32694		"ddd". The precision specifies the minimum number of digits to appear; if the value
32695		being converted can be represented in fewer digits, it shall be expanded with leading
32696		zeros. The default precision shall be 1. The result of converting zero with an explicit
32697		precision of zero shall be no wide characters.
32698	x	The unsigned argument shall be converted to unsigned hexadecimal format in the style
32699		"ddd"; the letters "abcdef" are used. The precision specifies the minimum number
32700		of digits to appear; if the value being converted can be represented in fewer digits, it
32701		shall be expanded with leading zeros. The default precision shall be 1. The result of
32702		converting zero with an explicit precision of zero shall be no wide characters.
32703	X	Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead
32704		of "abcdef".
32705	f, F	The double argument shall be converted to decimal notation in the style
32706		"[−] ddd.ddd", where the number of digits after the radix character shall be equal to
32707		the precision specification. If the precision is missing, it shall be taken as 6; if the
32708		precision is explicitly zero and no '#' flag is present, no radix character shall appear. If
32709		a radix character appears, at least one digit shall appear before it. The value shall be
32710		rounded in an implementation-defined manner to the appropriate number of digits.
32711		A double argument representing an infinity shall be converted in one of the styles
32712		"[−]inf" or "[−]infinity"; which style is implementation-defined. A double
32713		argument representing a NaN shall be converted in one of the styles "[−]nan" or
32714		"[−]nan(n-char-sequence)"; which style, and the meaning of any n-char-sequence,
32715		is implementation-defined. The F conversion specifier produces "INF", "INFINITY",
32716		or "NAN" instead of "inf", "infinity", or "nan", respectively.
32717	e, E	The double argument shall be converted in the style "[−] d.ddde±dd", where there
32718		shall be one digit before the radix character (which is non-zero if the argument is non-
32719		zero) and the number of digits after it shall be equal to the precision; if the precision is
32720		missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no
32721		radix character shall appear. The value shall be rounded in an implementation-defined
32722		manner to the appropriate number of digits. The E conversion wide character shall
32723		produce a number with 'E' instead of 'e' introducing the exponent. The exponent
32724		shall always contain at least two digits. If the value is zero, the exponent shall be zero.
32725		A double argument representing an infinity or NaN shall be converted in the style of
32726		an f or F conversion specifier.

32727	g, G	The double argument representing a floating-point number shall be converted in the style f or e (or in the style F or E in the case of a G conversion specifier), depending on the value converted and the precision. Let P equal the precision if non-zero, 6 if the precision is omitted, or 1 if the precision is zero. Then, if a conversion with style E would have an exponent of X :
32728		
32729		
32730		
32731		
32732		— If $P > X \geq -4$, the conversion shall be with style f (or F) and precision $P - (X + 1)$.
32733		— Otherwise, the conversion shall be with style e (or E) and precision $P - 1$.
32734		Finally, unless the '#' flag is used, any trailing zeros shall be removed from the fractional portion of the result and the decimal-point character shall be removed if there is no fractional portion remaining.
32735		
32736		
32737		A double argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.
32738		
32739	a, A	A double argument representing a floating-point number shall be converted in the style "[−]0xh.hhhhp±d", where there shall be one hexadecimal digit (which is non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point wide character and the number of hexadecimal digits after it shall be equal to the precision; if the precision is missing and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and FLT_RADIX is not a power of 2, then the precision shall be sufficient to distinguish values of type double , except that trailing zeros may be omitted; if the precision is zero and the '#' flag is not specified, no decimal-point wide character shall appear. The letters "abcdef" are used for a conversion and the letters "ABCDEF" for A conversion. The A conversion specifier produces a number with 'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero.
32740		
32741		
32742		
32743		
32744		
32745		
32746		
32747		
32748		
32749		
32750		
32751		
32752		
32753		
32754		
32755	c	If no l (ell) qualifier is present, the int argument shall be converted to a wide character as if by calling the <i>btowc()</i> function and the resulting wide character shall be written. Otherwise, the wint_t argument shall be converted to wchar_t , and written.
32756		
32757		
32758	s	If no l (ell) qualifier is present, the application shall ensure that the argument is a pointer to a character array containing a character sequence beginning in the initial shift state. Characters from the array shall be converted as if by repeated calls to the <i>mbrtowc()</i> function, with the conversion state described by an mbstate_t object initialized to zero before the first character is converted, and written up to (but not including) the terminating null wide character. If the precision is specified, no more than that many wide characters shall be written. If the precision is not specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character.
32759		
32760		
32761		
32762		
32763		
32764		
32765		
32766		
32767		
32768		
32769		
32770		
32771		
32772		

fwprintf()*System Interfaces*

32773	p	The application shall ensure that the argument is a pointer to void . The value of the pointer shall be converted to a sequence of printable wide characters in an implementation-defined manner.
32774		
32775		
32776	n	The application shall ensure that the argument is a pointer to an integer into which is written the number of wide characters written to the output so far by this call to one of the <i>fwprintf()</i> functions. No argument shall be converted, but one shall be consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.
32777		
32778		
32779		
32780		
32781	XSI	C Equivalent to lc .
32782	XSI	S Equivalent to ls .
32783	%	Output a ' % ' wide character; no argument shall be converted. The entire conversion specification shall be %%.
32784		
32785		If a conversion specification does not match one of the above forms, the behavior is undefined.
32786		In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by <i>fwprintf()</i> and <i>wprintf()</i> shall be printed as if <i>fputwc()</i> had been called.
32787		
32788		
32789		
32790		For a and A conversions, if FLT_RADIX is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.
32791		
32792		
32793		
32794		For e, E, f, F, g, and G conversion specifiers, if the number of significant decimal digits is at most DECIMAL_DIG, then the result should be correctly rounded. If the number of significant decimal digits is more than DECIMAL_DIG but the source value is exactly representable with DECIMAL_DIG digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having DECIMAL_DIG significant digits; the value of the resultant decimal string D should satisfy $L \leq D \leq U$, with the extra stipulation that the error should have a correct sign for the current rounding direction.
32795		
32796		
32797		
32798		
32799		
32800		
32801		
32802	CX	The last data modification and last file status change timestamps of the file shall be marked for update between the call to a successful execution of <i>fwprintf()</i> or <i>wprintf()</i> and the next successful completion of a call to <i>fflush()</i> or <i>fclose()</i> on the same stream, or a call to <i>exit()</i> or <i>abort()</i> .
32803		
32804		
32805		
32806	RETURN VALUE	
32807		Upon successful completion, these functions shall return the number of wide characters transmitted, excluding the terminating null wide character in the case of <i>swprintf()</i> , or a negative value if an output error was encountered, and set <i>errno</i> to indicate the error.
32808		
32809	CX	
32810		If <i>n</i> or more wide characters were requested to be written, <i>swprintf()</i> shall return a negative value, and set <i>errno</i> to indicate the error.
32811	CX	
32812	ERRORS	
32813		For the conditions under which <i>fwprintf()</i> and <i>wprintf()</i> fail and may fail, refer to <i>fputwc()</i> .
32814		In addition, all forms of <i>fwprintf()</i> shall fail if:
32815	CX	[EILSEQ] A wide-character code that does not correspond to a valid character has been detected.
32816		

32817 In addition, all forms of *fwprintf()* may fail if:

32818 CX [EINVAL] There are insufficient arguments.

32819 In addition, *fwprintf()* and *wprintf()* may fail if:

32820 CX [ENOMEM] Insufficient storage space is available.

32821 The *swprintf()* shall fail if:

32822 CX [EOVERFLOW] The value of *n* is greater than {INT_MAX} or the number of bytes needed to
32823 hold the output excluding the terminating null is greater than {INT_MAX}.

32824 EXAMPLES

32825 To print the language-independent date and time format, the following statement could be used:

32826 `wprintf(format, weekday, month, day, hour, min);`

32827 For American usage, *format* could be a pointer to the wide-character string:

32828 `L"%s, %s %d, %d:%.2d\n"`

32829 producing the message:

32830 `Sunday, July 3, 10:02`

32831 whereas for German usage, *format* could be a pointer to the wide-character string:

32832 `L"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"`

32833 producing the message:

32834 `Sonntag, 3. Juli, 10:02`

32835 APPLICATION USAGE

32836 None.

32837 RATIONALE

32838 None.

32839 FUTURE DIRECTIONS

32840 None.

32841 SEE ALSO

32842 *btowc()*, *fputwc()*, *fwscanf()*, *mbrtowc()*, *setlocale()*

32843 XBD Chapter 7 (on page 135), `<stdio.h>`, `<wchar.h>`

32844 CHANGE HISTORY

32845 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
32846 (E).

32847 Issue 6

32848 The Open Group Corrigendum U040/1 is applied to the RETURN VALUE section, describing
32849 the case if *n* or more wide characters are requested to be written using *swprintf()*.

32850 The normative text is updated to avoid use of the term “must” for application requirements.

32851 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 32852 • The prototypes for *fwprintf()*, *swprintf()*, and *wprintf()* are updated.
- 32853 • The DESCRIPTION is updated.

fwprintf()*System Interfaces*

- 32854 • The `hh`, `ll`, `j`, `t`, and `z` length modifiers are added.
- 32855 • The `a`, `A`, and `F` conversion characters are added.
- 32856 • XSI shading is removed from the description of character string representations of infinity and NaN floating-point values.
- 32857
- 32858 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion specification” consistently.
- 32859
- 32860 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.
- 32861 **Issue 7**
- 32862 Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the `0` flag.
- 32863
- 32864 Austin Group Interpretation 1003.1-2001 #170 is applied.
- 32865 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied, revising the description of `g` and `G`.
- 32866
- 32867 Functionality relating to the “`%n$`” form of conversion specification and the `<apostrophe>` flag is moved from the XSI option to the Base.
- 32868
- 32869 The [EOVERFLOW] error is added for `swprintf()`.
- 32870 Changes are made related to support for finegrained timestamps.

NAME

fwrite — binary output

SYNOPSIS

#include <stdio.h>

```
size_t fwrite(const void *restrict ptr, size_t size, size_t nitems,
FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *fwrite()* function shall write, from the array pointed to by *ptr*, up to *nitems* elements whose size is specified by *size*, to the stream pointed to by *stream*. For each object, *size* calls shall be made to the *fputc()* function, taking the values (in order) from an array of **unsigned char** exactly overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by the number of bytes successfully written. If an error occurs, the resulting value of the file-position indicator for the stream is unspecified.

CX The last data modification and last file status change timestamps of the file shall be marked for update between the successful execution of *fwrite()* and the next successful completion of a call to *fflush()* or *fclose()* on the same stream, or a call to *exit()* or *abort()*.

RETURN VALUE

The *fwrite()* function shall return the number of elements successfully written, which may be less than *nitems* if a write error is encountered. If *size* or *nitems* is 0, *fwrite()* shall return 0 and the state of the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for the stream shall be set, and *errno* shall be set to indicate the error.

ERRORSRefer to *fputc()*.**EXAMPLES**

None.

APPLICATION USAGE

Because of possible differences in element length and byte ordering, files written using *fwrite()* are application-dependent, and possibly cannot be read using *fread()* by a different application or by the same application on a different processor.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO*ferror()*, *fopen()*, *fprintf()*, *putc()*, *puts()*, *write()*

XBD <stdio.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

fwrite()*System Interfaces*32912 **Issue 6**

32913 Extensions beyond the ISO C standard are marked.

32914 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 32915 • The *fwrite()* prototype is updated.
- 32916 • The DESCRIPTION is updated to clarify how the data is written out using *fputc()*.

32917 **Issue 7**

32918 Changes are made related to support for finegrained timestamps.

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

System Interfaces

fwscanf()**NAME**

fwscanf, swscanf, wscanf — convert formatted wide-character input

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int fwscanf(FILE *restrict stream, const wchar_t *restrict format, ...);
int swscanf(const wchar_t *restrict ws,
            const wchar_t *restrict format, ...);
int wscanf(const wchar_t *restrict format, ...);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *fwscanf()* function shall read from the named input *stream*. The *wscanf()* function shall read from the standard input stream *stdin*. The *swscanf()* function shall read from the wide-character string *ws*. Each function reads wide characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control wide-character string *format* described below, and a set of *pointer* arguments indicating where the converted input should be stored. The result is undefined if there are insufficient arguments for the format. If the *format* is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier wide character % (see below) is replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}]. This feature provides for the definition of *format* wide-character strings that select arguments in an order appropriate to specific languages. In *format* wide-character strings containing the "%n\$" form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the *format* wide-character string more than once.

The *format* can contain either form of a conversion specification—that is, % or "%n\$"—but the two forms cannot normally be mixed within a single *format* wide-character string. The only exception to this is that %% or %* can be mixed with the "%n\$" form. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*−1)th, are pointers.

The *fwscanf()* function in all its forms allows for detection of a language-dependent radix character in the input string, encoded as a wide-character value. The radix character is defined in the locale of the process (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> ('.').

The *format* is a wide-character string composed of zero or more directives. Each directive is composed of one of the following: one or more white-space wide characters (<space>, <tab>, <newline>, <vertical-tab>, or <form-feed>); an ordinary wide character (neither '%' nor a white-space character); or a conversion specification.

Each conversion specification is introduced by the '%' or by the character sequence "%n\$", after which the following appear in sequence:

- An optional assignment-suppressing character '*'.

fwscanf()*System Interfaces*

- 32964
- An optional non-zero decimal integer that specifies the maximum field width.
- 32965 CX
- An optional assignment-allocation character 'm'.
- 32966
- An optional length modifier that specifies the size of the receiving object.
- 32967
- A conversion specifier wide character that specifies the type of conversion to be applied.
- 32968
- The valid conversion specifiers are described below.
- 32969
- The *fwscanf()* functions shall execute each directive of the format in turn. If a directive fails, as detailed below, the function shall return. Failures are described as input failures (due to the unavailability of input bytes) or matching failures (due to inappropriate input).
- 32970
- 32971
- 32972
- A directive composed of one or more white-space wide characters is executed by reading input until no more valid input can be read, or up to the first wide character which is not a white-space wide character, which remains unread.
- 32973
- 32974
- 32975
- A directive that is an ordinary wide character shall be executed as follows. The next wide character is read from the input and compared with the wide character that comprises the directive; if the comparison shows that they are not equivalent, the directive shall fail, and the differing and subsequent wide characters remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a wide character from being read, the directive shall fail.
- 32976
- 32977
- 32978
- 32979
- 32980
- A directive that is a conversion specification defines a set of matching input sequences, as described below for each conversion wide character. A conversion specification is executed in the following steps.
- 32981
- 32982
- 32983
- Input white-space wide characters (as specified by *iswspace()*) shall be skipped, unless the conversion specification includes a *l*, *c*, or *n* conversion specifier.
- 32984
- 32985
- An item shall be read from the input, unless the conversion specification includes an *n* conversion specifier wide character. An input item is defined as the longest sequence of input wide characters, not exceeding any specified field width, which is an initial subsequence of a matching sequence. The first wide character, if any, after the input item shall remain unread. If the length of the input item is zero, the execution of the conversion specification shall fail; this condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.
- 32986
- 32987
- 32988
- 32989
- 32990
- 32991
- 32992
- Except in the case of a *%* conversion specifier, the input item (or, in the case of a *%n* conversion specification, the count of input wide characters) shall be converted to a type appropriate to the conversion wide character. If the input item is not a matching sequence, the execution of the conversion specification shall fail; this condition is a matching failure. Unless assignment suppression was indicated by a *'*'*, the result of the conversion shall be placed in the object pointed to by the first argument following the *format* argument that has not already received a conversion result if the conversion specification is introduced by *%*, or in the *n*th argument if introduced by the wide-character sequence *"%n\$"*. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.
- 32993
- 32994
- 32995
- 32996
- 32997
- 32998 CX
- 32999
- 33000
- 33001
- 33002 CX
- 33003
- 33004
- 33005
- 33006
- 33007
- 33008
- 33009
- The *%c*, *%s*, and *%[* conversion specifiers shall accept an optional assignment-allocation character 'm', which shall cause a memory buffer to be allocated to hold the wide-character string converted including a terminating null wide character. In such a case, the argument corresponding to the conversion specifier should be a reference to a pointer value that will receive a pointer to the allocated buffer. The system shall allocate a buffer as if *malloc()* had been called. The application shall be responsible for freeing the memory after usage. If there is insufficient memory to allocate a buffer, the function shall set *errno* to [ENOMEM] and a conversion error shall result. If the function returns EOF, any memory successfully allocated for

33010		parameters using assignment-allocation character 'm' by this call shall be freed before the
33011		function returns.
33012		The length modifiers and their meanings are:
33013	hh	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
33014		argument with type pointer to signed char or unsigned char .
33015	h	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
33016		argument with type pointer to short or unsigned short .
33017	l (ell)	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
33018		argument with type pointer to long or unsigned long ; that a following a, A, e, E, f, F,
33019		g, or G conversion specifier applies to an argument with type pointer to double ; or that
33020		a following c, s, or l conversion specifier applies to an argument with type pointer to
33021	CX	wchar_t . If the 'm' assignment-allocation character is specified, the conversion
33022		applies to an argument with the type pointer to a pointer to wchar_t .
33023	ll (ell-ell)	
33024		Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
33025		argument with type pointer to long long or unsigned long long .
33026	j	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
33027		argument with type pointer to intmax_t or uintmax_t .
33028	z	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
33029		argument with type pointer to size_t or the corresponding signed integer type.
33030	t	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
33031		argument with type pointer to ptrdiff_t or the corresponding unsigned type.
33032	L	Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an
33033		argument with type pointer to long double .
33034		If a length modifier appears with any conversion specifier other than as specified above, the
33035		behavior is undefined.
33036		The following conversion specifier wide characters are valid:
33037	d	Matches an optionally signed decimal integer, whose format is the same as expected for
33038		the subject sequence of <i>wcstol()</i> with the value 10 for the <i>base</i> argument. In the absence
33039		of a size modifier, the application shall ensure that the corresponding argument is a
33040		pointer to int .
33041	i	Matches an optionally signed integer, whose format is the same as expected for the
33042		subject sequence of <i>wcstol()</i> with 0 for the <i>base</i> argument. In the absence of a size
33043		modifier, the application shall ensure that the corresponding argument is a pointer to
33044		int .
33045	o	Matches an optionally signed octal integer, whose format is the same as expected for
33046		the subject sequence of <i>wcstoul()</i> with the value 8 for the <i>base</i> argument. In the absence
33047		of a size modifier, the application shall ensure that the corresponding argument is a
33048		pointer to unsigned .
33049	u	Matches an optionally signed decimal integer, whose format is the same as expected for
33050		the subject sequence of <i>wcstoul()</i> with the value 10 for the <i>base</i> argument. In the absence
33051		of a size modifier, the application shall ensure that the corresponding argument is a
33052		pointer to unsigned .

fwscanf()*System Interfaces*

33053	x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <i>wcstoul()</i> with the value 16 for the <i>base</i> argument.
33054		In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
33055		
33056		
33057	a, e, f, g	Matches an optionally signed floating-point number, infinity, or NaN whose format is the same as expected for the subject sequence of <i>wcstod()</i> . In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to float .
33058		
33059		
33060		
33061		
33062		If the <i>fwprintf()</i> family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std 754-1985, the <i>fwscanf()</i> family of functions shall recognize them as input.
33063		
33064		
33065	s	Matches a sequence of non-white-space wide characters. If no l (ell) qualifier is present, characters from the input field shall be converted as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by an mbstate_t object initialized to zero before the first wide character is converted. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to a character array large enough to accept the sequence and the terminating null character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a wchar_t .
33066		
33067		
33068		
33069		
33070		
33071	CX	
33072		
33073		
33074		If the l (ell) qualifier is present and the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of wchar_t large enough to accept the sequence and the terminating null wide character, which shall be added automatically. If the l (ell) qualifier is present and the 'm' assignment-allocation character is present, the application shall ensure that the corresponding argument is a pointer to a pointer to a wchar_t .
33075		
33076		
33077	CX	
33078		
33079		
33080	[	Matches a non-empty sequence of wide characters from a set of expected wide characters (the <i>scanlist</i>). If no l (ell) qualifier is present, wide characters from the input field shall be converted as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by an mbstate_t object initialized to zero before the first wide character is converted. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to a character array large enough to accept the sequence and the terminating null character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a wchar_t .
33081		
33082		
33083		
33084		
33085		
33086		
33087	CX	
33088		
33089		If an l (ell) qualifier is present and the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of wchar_t large enough to accept the sequence and the terminating null wide character. If an l (ell) qualifier is present and the 'm' assignment-allocation character is specified, the application shall ensure that the corresponding argument is a pointer to a pointer to a wchar_t .
33090		
33091		
33092	CX	
33093		
33094		
33095		The conversion specification includes all subsequent wide characters in the <i>format</i> string up to and including the matching <right-square-bracket> ('] '). The wide characters between the square brackets (the <i>scanlist</i>) comprise the <i>scanset</i> , unless the wide character after the <left-square-bracket> is a <circumflex> (' ^ '), in which case the <i>scanset</i> contains all wide characters that do not appear in the <i>scanlist</i> between the <circumflex> and the <right-square-bracket>. If the conversion specification begins
33096		
33097		
33098		
33099		
33100		

33101		with "[]" or "[^]", the <right-square-bracket> is included in the scanlist and the
33102		next <right-square-bracket> is the matching <right-square-bracket> that ends the
33103		conversion specification; otherwise, the first <right-square-bracket> is the one that ends
33104		the conversion specification. If a '-' is in the scanlist and is not the first wide character,
33105		nor the second where the first wide character is a '^', nor the last wide character, the
33106		behavior is implementation-defined.
33107	c	Matches a sequence of wide characters of exactly the number specified by the field
33108		width (1 if no field width is present in the conversion specification).
33109		If no l (ell) length modifier is present, characters from the input field shall be converted
33110		as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by
33111		an mbstate_t object initialized to zero before the first wide character is converted. No
33112		null character is added. If the 'm' assignment-allocation character is not specified, the
33113		application shall ensure that the corresponding argument is a pointer to the initial
33114	CX	element of a character array large enough to accept the sequence. Otherwise, the
33115		application shall ensure that the corresponding argument is a pointer to a pointer to a
33116		char .
33117		No null wide character is added. If an l (ell) length modifier is present and the 'm'
33118		assignment-allocation character is not specified, the application shall ensure that the
33119		corresponding argument shall be a pointer to the initial element of an array of wchar_t
33120	CX	large enough to accept the sequence. If an l (ell) qualifier is present and the 'm'
33121		assignment-allocation character is specified, the application shall ensure that the
33122		corresponding argument is a pointer to a pointer to a wchar_t .
33123	p	Matches an implementation-defined set of sequences, which shall be the same as the set
33124		of sequences that is produced by the %p conversion specification of the corresponding
33125		<i>fwprintf()</i> functions. The application shall ensure that the corresponding argument is a
33126		pointer to a pointer to void . The interpretation of the input item is implementation-
33127		defined. If the input item is a value converted earlier during the same program
33128		execution, the pointer that results shall compare equal to that value; otherwise, the
33129		behavior of the %p conversion is undefined.
33130	n	No input is consumed. The application shall ensure that the corresponding argument is
33131		a pointer to the integer into which is to be written the number of wide characters read
33132		from the input so far by this call to the <i>fwscanf()</i> functions. Execution of a %n
33133		conversion specification shall not increment the assignment count returned at the
33134		completion of execution of the function. No argument shall be converted, but one shall
33135		be consumed. If the conversion specification includes an assignment-suppressing wide
33136		character or a field width, the behavior is undefined.
33137	XSI	C Equivalent to lc.
33138	XSI	S Equivalent to ls.
33139	%	Matches a single '%' wide character; no conversion or assignment shall occur. The
33140		complete conversion specification shall be %.
33141		If a conversion specification is invalid, the behavior is undefined.
33142		The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to, respectively,
33143		a, e, f, g, and x.
33144		If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before
33145		any wide characters matching the current conversion specification (except for %n) have been
33146		read (other than leading white-space, where permitted), execution of the current conversion

fwscanf()

System Interfaces

specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.

Reaching the end of the string in *swscanf()* shall be equivalent to encountering end-of-file for *fwscanf()*.

If conversion terminates on a conflicting input, the offending input shall be left unread in the input. Any trailing white space (including <newline>) shall be left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the %n conversion specification.

CX The *fwscanf()* and *wscanf()* functions may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetwc()*, *fgetws()*, *fwscanf()*, *getwc()*, *getwchar()*, *vfwscanf()*, *vwscanf()*, or *wscanf()* using *stream* that returns data not supplied by a prior call to *ungetwc()*.

RETURN VALUE

Upon successful completion, these functions shall return the number of successfully matched and assigned input items; this number can be zero in the event of an early matching failure. If the input ends before the first matching failure or conversion, EOF shall be returned. If any error occurs, EOF shall be returned, and *errno* shall be set to indicate the error. If a read error occurs, the error indicator for the stream shall be set.

ERRORS

For the conditions under which the *fwscanf()* functions shall fail and may fail, refer to *fgetwc()*.

In addition, the *fwscanf()* function shall fail if:

CX [EILSEQ] Input byte sequence does not form a valid character.

[ENOMEM] Insufficient storage space is available.

In addition, the *fwscanf()* function may fail if:

CX [EINVAL] There are insufficient arguments.

EXAMPLES

The call:

```
int i, n; float x; char name[50];
n = wscanf(L"%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 Hamster
```

assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string "Hamster".

The call:

```
int i; float x; char name[50];
(void) wscanf(L"%2d%f*d %[0123456789]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to *getchar()* shall return the character 'a'.

APPLICATION USAGE

In format strings containing the ' % ' form of conversion specifications, each argument in the argument list is used exactly once.

For functions that allocate memory as if by *malloc()*, the application should release such memory when it is no longer required by a call to *free()*. For *fwscanf()*, this is memory allocated via use of the ' m ' assignment-allocation character.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getwc(), *fwprintf()*, *setlocale()*, *wcstod()*, *wcstol()*, *wcstoul()*, *wcrtomb()*

XBD Chapter 7 (on page 135), *<stdio.h>*, *<wchar.h>*

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The prototypes for *fwscanf()* and *swscanf()* are updated.
- The DESCRIPTION is updated.
- The hh, ll, j, t, and z length modifiers are added.
- The a, A, and F conversion characters are added.

The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion specification” consistently.

Issue 7

Austin Group Interpretation 1003.1-2001 #170 is applied.

SD5-XSH-ERN-132 is applied, adding the assignment-allocation character ' m '.

Functionality relating to the "%n\$" form of conversion specification is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

The APPLICATION USAGE section is updated to clarify that memory is allocated as if by *malloc()*.

gai_strerror()

System Interfaces

33221 **NAME**33222 **gai_strerror** — address and name information error description33223 **SYNOPSIS**33224 `#include <netdb.h>`33225 `const char *gai_strerror(int ecode);`33226 **DESCRIPTION**33227 The *gai_strerror()* function shall return a text string describing an error value for the *getaddrinfo()*
33228 and *getnameinfo()* functions listed in the **<netdb.h>** header.33229 When the *ecode* argument is one of the following values listed in the **<netdb.h>** header:

33230 [EAL_AGAIN]	[EAL_NONAME]
33231 [EAL_BADFLAGS]	[EAL_OVERFLOW]
33232 [EAL_FAIL]	[EAL_SERVICE]
33233 [EAL_FAMILY]	[EAL_SOCKETTYPE]
33234 [EAL_MEMORY]	[EAL_SYSTEM]

33235 the function return value shall point to a string describing the error. If the argument is not one
33236 of those values, the function shall return a pointer to a string whose contents indicate an
33237 unknown error.

33238 **RETURN VALUE**

33239 Upon successful completion, *gai_strerror()* shall return a pointer to an implementation-defined
33240 string.

33241 **ERRORS**

33242 No errors are defined.

33243 **EXAMPLES**

33244 None.

33245 **APPLICATION USAGE**

33246 None.

33247 **RATIONALE**

33248 None.

33249 **FUTURE DIRECTIONS**

33250 None.

33251 **SEE ALSO**33252 *freeaddrinfo()*33253 XBD **<netdb.h>**33254 **CHANGE HISTORY**

33255 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

33256 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type from
33257 **char *** to **const char ***. This is for coordination with the IPnG Working Group.

33258 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/22 is applied, adding the
33259 [EAL_OVERFLOW] error code.

*System Interfaces***getaddrinfo()**33260 **NAME**

33261 getaddrinfo — get address information

33262 **SYNOPSIS**

```
33263       #include <sys/socket.h>
33264       #include <netdb.h>

33265       int getaddrinfo(const char *restrict nodename,
33266                      const char *restrict servname,
33267                      const struct addrinfo *restrict hints,
33268                      struct addrinfo **restrict res);
```

33269 **DESCRIPTION**33270 Refer to *freeaddrinfo()*.

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

getc()33271 **NAME**33272 `getc` — get a byte from a stream33273 **SYNOPSIS**33274 `#include <stdio.h>`33275 `int getc(FILE *stream);`33276 **DESCRIPTION**

33277 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 33278 conflict between the requirements described here and the ISO C standard is unintentional. This
 33279 volume of POSIX.1-2008 defers to the ISO C standard.

33280 The `getc()` function shall be equivalent to `fgetc()`, except that if it is implemented as a macro it
 33281 may evaluate *stream* more than once, so the argument should never be an expression with side-
 33282 effects.

33283 **RETURN VALUE**33284 Refer to `fgetc()`.33285 **ERRORS**33286 Refer to `fgetc()`.33287 **EXAMPLES**

33288 None.

33289 **APPLICATION USAGE**

33290 If the integer value returned by `getc()` is stored into a variable of type **char** and then compared
 33291 against the integer constant EOF, the comparison may never succeed, because sign-extension of
 33292 a variable of type **char** on widening to integer is implementation-defined.

33293 Since it may be implemented as a macro, `getc()` may treat incorrectly a *stream* argument with
 33294 side-effects. In particular, `getc(*f++)` does not necessarily work as expected. Therefore, use of this
 33295 function should be preceded by `#undef getc` in such situations; `fgetc()` could also be used.

33296 **RATIONALE**

33297 None.

33298 **FUTURE DIRECTIONS**

33299 None.

33300 **SEE ALSO**33301 `fgetc()`33302 XBD `<stdio.h>`33303 **CHANGE HISTORY**

33304 First released in Issue 1. Derived from Issue 1 of the SVID.

NAME

`getc_unlocked`, `getchar_unlocked`, `putc_unlocked`, `putchar_unlocked` — `stdio` with explicit client locking

SYNOPSIS

```
CX      #include <stdio.h>

      int getc_unlocked(FILE *stream);
      int getchar_unlocked(void);
      int putc_unlocked(int c, FILE *stream);
      int putchar_unlocked(int c);
```

DESCRIPTION

Versions of the functions `getc()`, `getchar()`, `putc()`, and `putchar()` respectively named `getc_unlocked()`, `getchar_unlocked()`, `putc_unlocked()`, and `putchar_unlocked()` shall be provided which are functionally equivalent to the original versions, with the exception that they are not required to be implemented in a thread-safe manner. They may only safely be used within a scope protected by `flockfile()` (or `ftrylockfile()`) and `funlockfile()`. These functions may safely be used in a multi-threaded program if and only if they are called while the invoking thread owns the (FILE *) object, as is the case after a successful call to the `flockfile()` or `ftrylockfile()` functions.

RETURN VALUE

See `getc()`, `getchar()`, `putc()`, and `putchar()`.

ERRORS

See `getc()`, `getchar()`, `putc()`, and `putchar()`.

EXAMPLES

None.

APPLICATION USAGE

Since they may be implemented as macros, `getc_unlocked()` and `putc_unlocked()` may treat incorrectly a `stream` argument with side-effects. In particular, `getc_unlocked(*f++)` and `putc_unlocked(*f++)` do not necessarily work as expected. Therefore, use of these functions in such situations should be preceded by the following statement as appropriate:

```
#undef getc_unlocked
#undef putc_unlocked
```

RATIONALE

Some I/O functions are typically implemented as macros for performance reasons (for example, `putc()` and `getc()`). For safety, they need to be synchronized, but it is often too expensive to synchronize on every character. Nevertheless, it was felt that the safety concerns were more important; consequently, the `getc()`, `getchar()`, `putc()`, and `putchar()` functions are required to be thread-safe. However, unlocked versions are also provided with names that clearly indicate the unsafe nature of their operation but can be used to exploit their higher performance. These unlocked versions can be safely used only within explicitly locked program regions, using exported locking primitives. In particular, a sequence such as:

```
flockfile(fileptr);
putc_unlocked('1', fileptr);
putc_unlocked('\n', fileptr);
fprintf(fileptr, "Line 2\n");
funlockfile(fileptr);
```

getc_unlocked()*System Interfaces*

is permissible, and results in the text sequence:

1
Line 2

being printed without being interspersed with output from other threads.

It would be wrong to have the standard names such as *getc()*, *putc()*, and so on, map to the “faster, but unsafe” rather than the “slower, but safe” versions. In either case, you would still want to inspect all uses of *getc()*, *putc()*, and so on, by hand when converting existing code. Choosing the safe bindings as the default, at least, results in correct code and maintains the “atomicity at the function” invariant. To do otherwise would introduce gratuitous synchronization errors into converted code. Other routines that modify the *stdio* (FILE *) structures or buffers are also safely synchronized.

Note that there is no need for functions of the form *getc_locked()*, *putc_locked()*, and so on, since this is the functionality of *getc()*, *putc()*, *et al.* It would be inappropriate to use a feature test macro to switch a macro definition of *getc()* between *getc_locked()* and *getc_unlocked()*, since the ISO C standard requires an actual function to exist, a function whose behavior could not be changed by the feature test macro. Also, providing both the *xxx_locked()* and *xxx_unlocked()* forms leads to the confusion of whether the suffix describes the behavior of the function or the circumstances under which it should be used.

Three additional routines, *flockfile()*, *ftrylockfile()*, and *funlockfile()* (which may be macros), are provided to allow the user to delineate a sequence of I/O statements that are executed synchronously.

The *ungetc()* function is infrequently called relative to the other functions/macros so no unlocked variation is needed.

FUTURE DIRECTIONS

None.

SEE ALSO

getc(), *getchar()*, *putc()*, *putchar()*

XBD <stdio.h>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

These functions are marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U030/2 is applied, adding APPLICATION USAGE describing how applications should be written to avoid the case when the functions are implemented as macros.

Issue 7

The *getc_unlocked()*, *getchar_unlocked()*, *putc_unlocked()*, and *putchar_unlocked()* functions are moved from the Thread-Safe Functions option to the Base.

NAME

getchar — get a byte from a *stdin* stream

SYNOPSIS

```
#include <stdio.h>
```

```
int getchar(void);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *getchar()* function shall be equivalent to *getc(stdin)*.

RETURN VALUE

Refer to *fgetc()*.

ERRORS

Refer to *fgetc()*.

EXAMPLES

None.

APPLICATION USAGE

If the integer value returned by *getchar()* is stored into a variable of type **char** and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type **char** on widening to integer is implementation-defined.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getc()

XBD *<stdio.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

getchar_unlocked()*System Interfaces*33416 **NAME**

33417 getchar_unlocked — stdio with explicit client locking

33418 **SYNOPSIS**

```
33419 CX       #include <stdio.h>  
33420       int getchar_unlocked(void);
```

33421 **DESCRIPTION**33422 Refer to *getc_unlocked()*.

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

System Interfaces

getcwd()**NAME**

getcwd — get the pathname of the current working directory

SYNOPSIS

```
#include <unistd.h>

char *getcwd(char *buf, size_t size);
```

DESCRIPTION

The *getcwd()* function shall place an absolute pathname of the current working directory in the array pointed to by *buf*, and return *buf*. The pathname shall contain no components that are dot or dot-dot, or are symbolic links.

If there are multiple pathnames that *getcwd()* could place in the array pointed to by *buf*, one beginning with a single <slash> character and one or more beginning with two <slash> characters, then *getcwd()* shall place the pathname beginning with a single <slash> character in the array. The pathname shall not contain any unnecessary <slash> characters after the leading one or two <slash> characters.

The *size* argument is the size in bytes of the character array pointed to by the *buf* argument. If *buf* is a null pointer, the behavior of *getcwd()* is unspecified.

RETURN VALUE

Upon successful completion, *getcwd()* shall return the *buf* argument. Otherwise, *getcwd()* shall return a null pointer and set *errno* to indicate the error. The contents of the array pointed to by *buf* are then undefined.

ERRORS

The *getcwd()* function shall fail if:

[EINVAL]	The <i>size</i> argument is 0.
[ERANGE]	The <i>size</i> argument is greater than 0, but is smaller than the length of the string +1.

The *getcwd()* function may fail if:

[EACCES]	Search permission was denied for the current directory, or read or search permission was denied for a directory above the current directory in the file hierarchy.
[ENOMEM]	Insufficient storage space is available.

EXAMPLES

The following example uses {PATH_MAX} as the initial buffer size (unless it is indeterminate or very large), and calls *getcwd()* with progressively larger buffers until it does not give an [ERANGE] error.

```
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

...

long path_max;
size_t size;
char *buf;
char *ptr;

path_max = pathconf(".", _PC_PATH_MAX);
if (path_max == -1)
```

getcwd()

```

33467         size = 1024;
33468     else if (path_max > 10240)
33469         size = 10240;
33470     else
33471         size = path_max;
33472     for (buf = ptr = NULL; ptr == NULL; size *= 2)
33473     {
33474         if ((buf = realloc(buf, size)) == NULL)
33475         {
33476             ... handle error ...
33477         }
33478         ptr = getcwd(buf, size);
33479         if (ptr == NULL && errno != ERANGE)
33480         {
33481             ... handle error ...
33482         }
33483     }
33484     free (buf);

```

APPLICATION USAGE

If the pathname obtained from *getcwd()* is longer than {PATH_MAX} bytes, it could produce an [ENAMETOOLONG] error if passed to *chdir()*. Therefore, in order to return to that directory it may be necessary to break the pathname into sections shorter than {PATH_MAX} bytes and call *chdir()* on each section in turn (the first section being an absolute pathname and subsequent sections being relative pathnames). A simpler way to handle saving and restoring the working directory when it may be deeper than {PATH_MAX} bytes in the file hierarchy is to use a file descriptor and *fchdir()*, rather than *getcwd()* and *chdir()*. However, the two methods do have some differences. The *fchdir()* approach causes the program to restore a working directory even if it has been renamed in the meantime, whereas the *chdir()* approach restores to a directory with the same name as the original, even if the directories were renamed in the meantime. Since the *fchdir()* approach does not access parent directories, it can succeed when *getcwd()* would fail due to permissions problems. In applications conforming to earlier versions of this standard, it was not possible to use the *fchdir()* approach when the working directory is searchable but not readable, as the only way to open a directory was with O_RDONLY, whereas the *getcwd()* approach can succeed in this case.

RATIONALE

Having *getcwd()* take no arguments and instead use the *malloc()* function to produce space for the returned argument was considered. The advantage is that *getcwd()* knows how big the working directory pathname is and can allocate an appropriate amount of space. But the programmer would have to use the *free()* function to free the resulting object, or each use of *getcwd()* would further reduce the available memory. Finally, *getcwd()* is taken from the SVID where it has the two arguments used in this volume of POSIX.1-2008.

The older function *getwd()* was rejected for use in this context because it had only a buffer argument and no *size* argument, and thus had no way to prevent overwriting the buffer, except to depend on the programmer to provide a large enough buffer.

On some implementations, if *buf* is a null pointer, *getcwd()* may obtain *size* bytes of memory using *malloc()*. In this case, the pointer returned by *getcwd()* may be used as the argument in a subsequent call to *free()*. Invoking *getcwd()* with *buf* as a null pointer is not recommended in conforming applications.

Earlier implementations of `getcwd()` sometimes generated pathnames like `"../../../../subdirname"` internally, using them to explore the path of ancestor directories back to the root. If one of these internal pathnames exceeded `{PATH_MAX}` in length, the implementation could fail with `errno` set to `[ENAMETOOLONG]`. This is no longer allowed.

If a program is operating in a directory where some (grand)parent directory does not permit reading, `getcwd()` may fail, as in most implementations it must read the directory to determine the name of the file. This can occur if search, but not read, permission is granted in an intermediate directory, or if the program is placed in that directory by some more privileged process (for example, login). Including the `[EACCES]` error condition makes the reporting of the error consistent and warns the application developer that `getcwd()` can fail for reasons beyond the control of the application developer or user. Some implementations can avoid this occurrence (for example, by implementing `getcwd()` using `pwd`, where `pwd` is a set-user-root process), thus the error was made optional. Since this volume of POSIX.1-2008 permits the addition of other errors, this would be a common addition and yet one that applications could not be expected to deal with without this addition.

FUTURE DIRECTIONS

None.

SEE ALSO

`malloc()`

XBD `<unistd.h>`

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The `[ENOMEM]` optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #140 is applied, changing the text for consistency with the `pwd` utility, adding text to address the case where the current directory is deeper in the file hierarchy than `{PATH_MAX}` bytes, and adding the requirements relating to pathnames beginning with two `<slash>` characters.

getdate()*System Interfaces*33546 **NAME**33547 `getdate` — convert user format date and time33548 **SYNOPSIS**

```
33549 XSI    #include <time.h>
33550        struct tm *getdate(const char *string);
```

33551 **DESCRIPTION**

33552 The `getdate()` function shall convert a string representation of a date or time into a broken-down
 33553 time.

33554 The external variable or macro `getdate_err`, which has type **int**, is used by `getdate()` to return error
 33555 values. It is unspecified whether `getdate_err` is a macro or an identifier declared with external
 33556 linkage, and whether or not it is a modifiable lvalue. If a macro definition is suppressed in order
 33557 to access an actual object, or a program defines an identifier with the name `getdate_err`, the
 33558 behavior is undefined.

33559 Templates are used to parse and interpret the input string. The templates are contained in a text
 33560 file identified by the environment variable `DATMSK`. The `DATMSK` variable should be set to
 33561 indicate the full pathname of the file that contains the templates. The first line in the template
 33562 that matches the input specification is used for interpretation and conversion into the internal
 33563 time format.

33564 The following conversion specifications shall be supported:

33565	<code>%%</code>	Equivalent to <code>%</code> .
33566	<code>%a</code>	Abbreviated weekday name.
33567	<code>%A</code>	Full weekday name.
33568	<code>%b</code>	Abbreviated month name.
33569	<code>%B</code>	Full month name.
33570	<code>%c</code>	Locale's appropriate date and time representation.
33571	<code>%C</code>	Century number [00,99]; leading zeros are permitted but not required.
33572	<code>%d</code>	Day of month [01,31]; the leading 0 is optional.
33573	<code>%D</code>	Date as <code>%m/%d/%y</code> .
33574	<code>%e</code>	Equivalent to <code>%d</code> .
33575	<code>%h</code>	Abbreviated month name.
33576	<code>%H</code>	Hour [00,23].
33577	<code>%I</code>	Hour [01,12].
33578	<code>%m</code>	Month number [01,12].
33579	<code>%M</code>	Minute [00,59].
33580	<code>%n</code>	Equivalent to <code><newline></code> .
33581	<code>%p</code>	Locale's equivalent of either AM or PM.
33582	<code>%r</code>	The locale's appropriate representation of time in AM and PM notation. In the POSIX 33583 locale, this shall be equivalent to <code>%I:%M:%S %p</code> .

System Interfaces

getdate()

33584	%R	Time as %H:%M.
33585	%S	Seconds [00,60]. The range goes to 60 (rather than stopping at 59) to allow positive leap seconds to be expressed. Since leap seconds cannot be predicted by any algorithm, leap second data must come from some external source.
33586		
33587		
33588	%t	Equivalent to <tab>.
33589	%T	Time as %H:%M:%S.
33590	%w	Weekday number (Sunday = [0,6]).
33591	%x	Locale's appropriate date representation.
33592	%X	Locale's appropriate time representation.
33593	%y	Year within century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.
33594		
33595		
33596	Note:	It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)
33597		
33598		
33599	%Y	Year as "%cYY" (for example, 2001).
33600	%Z	Timezone name or no characters if no timezone exists. If the timezone supplied by %Z is not the timezone that <i>getdate()</i> expects, an invalid input specification error shall result. The <i>getdate()</i> function calculates an expected timezone based on information supplied to the function (such as the hour, day, and month).
33601		
33602		
33603		
33604		The match between the template and input specification performed by <i>getdate()</i> shall be case-insensitive.
33605		
33606		The month and weekday names can consist of any combination of upper and lowercase letters. The process can request that the input date or time specification be in a specific language by setting the <i>LC_TIME</i> category (see <i>setlocale()</i>).
33607		
33608		
33609		Leading zeros are not necessary for the descriptors that allow leading zeros. However, at most two digits are allowed for those descriptors, including leading zeros. Extra white space in either the template file or in <i>string</i> shall be ignored.
33610		
33611		
33612		The results are undefined if the conversion specifications %c, %x, and %X include unsupported conversion specifications.
33613		
33614		The following rules apply for converting the input specification into the internal format:
33615		• If %Z is being scanned, then <i>getdate()</i> shall initialize the broken-down time to be the current time in the scanned timezone. Otherwise, it shall initialize the broken-down time based on the current local time as if <i>localtime()</i> had been called.
33616		
33617		
33618		• If only the weekday is given, the day chosen shall be the day, starting with today and moving into the future, which first matches the named day.
33619		
33620		• If only the month (and no year) is given, the month chosen shall be the month, starting with the current month and moving into the future, which first matches the named month. The first day of the month shall be assumed if no day is given.
33621		
33622		
33623		• If no hour, minute, and second are given, the current hour, minute, and second shall be assumed.
33624		

getdate()*System Interfaces*

- 33625 • If no date is given, the hour chosen shall be the hour, starting with the current hour and
 33626 moving into the future, which first matches the named hour.

33627 If a conversion specification in the DATEMSK file does not correspond to one of the conversion
 33628 specifications above, the behavior is unspecified.

33629 The *getdate()* function need not be thread-safe.

RETURN VALUE

33630 Upon successful completion, *getdate()* shall return a pointer to a **struct tm**. Otherwise, it shall
 33631 return a null pointer and set *getdate_err* to indicate the error.
 33632

ERRORS

33633 The *getdate()* function shall fail in the following cases, setting *getdate_err* to the value shown in
 33634 the list below. Any changes to *errno* are unspecified.
 33635

- 33636 1. The DATEMSK environment variable is null or undefined.
- 33637 2. The template file cannot be opened for reading.
- 33638 3. Failed to get file status information.
- 33639 4. The template file is not a regular file.
- 33640 5. An I/O error is encountered while reading the template file.
- 33641 6. Memory allocation failed (not enough memory available).
- 33642 7. There is no line in the template that matches the input.
- 33643 8. Invalid input specification. For example, February 31; or a time is specified that cannot be
 33644 represented in a **time_t** (representing the time in seconds since the Epoch).

EXAMPLES

- 33645 1. The following example shows the possible contents of a template:
 33646

```
33647 %m
33648 %A %B %d, %Y, %H:%M:%S
33649 %A
33650 %B
33651 %m/%d/%Y %I %p
33652 %d, %m, %Y %H:%M
33653 at %A the %dst of %B in %Y
33654 run job at %I %p, %B %dnd
33655 %A den %d. %B %Y %H.%M Uhr
```

- 33656 2. The following are examples of valid input specifications for the template in Example 1:

```
33657 getdate("10/1/87 4 PM");
33658 getdate("Friday");
33659 getdate("Friday September 18, 1987, 10:30:30");
33660 getdate("24,9,1986 10:30");
33661 getdate("at monday the 1st of december in 1986");
33662 getdate("run job at 3 PM, december 2nd");
```

33663 If the *LC_TIME* category is set to a German locale that includes *freitag* as a weekday name
 33664 and *oktober* as a month name, the following would be valid:

```
33665 getdate("freitag den 10. oktober 1986 10.30 Uhr");
```

3. The following example shows how local date and time specification can be defined in the template:

Invocation	Line in Template
getdate("11/27/86")	%m/%d/%y
getdate("27.11.86")	%d.%m.%y
getdate("86-11-27")	%y-%m-%d
getdate("Friday 12:00:00")	%A %H:%M:%S

4. The following examples help to illustrate the above rules assuming that the current date is Mon Sep 22 12:19:47 EDT 1986 and the *LC_TIME* category is set to the default *C* locale:

Input	Line in Template	Date
Mon	%a	Mon Sep 22 12:19:47 EDT 1986
Sun	%a	Sun Sep 28 12:19:47 EDT 1986
Fri	%a	Fri Sep 26 12:19:47 EDT 1986
September	%B	Mon Sep 1 12:19:47 EDT 1986
January	%B	Thu Jan 1 12:19:47 EST 1987
December	%B	Mon Dec 1 12:19:47 EST 1986
Sep Mon	%b %a	Mon Sep 1 12:19:47 EDT 1986
Jan Fri	%b %a	Fri Jan 2 12:19:47 EST 1987
Dec Mon	%b %a	Mon Dec 1 12:19:47 EST 1986
Jan Wed 1989	%b %a %Y	Wed Jan 4 12:19:47 EST 1989
Fri 9	%a %H	Fri Sep 26 09:00:00 EDT 1986
Feb 10:30	%b %H:%S	Sun Feb 1 10:00:30 EST 1987
10:30	%H:%M	Tue Sep 23 10:30:00 EDT 1986
13:30	%H:%M	Mon Sep 22 13:30:00 EDT 1986

APPLICATION USAGE

Although historical versions of *getdate()* did not require that **<time.h>** declare the external variable *getdate_err*, this volume of POSIX.1-2008 does require it. The standard developers encourage applications to remove declarations of *getdate_err* and instead incorporate the declaration by including **<time.h>**.

Applications should use %Y (4-digit years) in preference to %y (2-digit years).

RATIONALE

In standard locales, the conversion specifications %C, %x, and %X do not include unsupported conversion specifiers and so the text regarding results being undefined is not a problem in that case.

FUTURE DIRECTIONS

None.

SEE ALSO

ctime(), *localtime()*, *setlocale()*, *strftime()*, *times()*

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

getdate()*System Interfaces***Issue 5**

- 33707 Moved from X/OPEN UNIX extension to BASE.
- 33708
- 33709 The last paragraph of the DESCRIPTION is added.
- 33710 The %C conversion specification is added, and the exact meaning of the %y conversion
- 33711 specification is clarified in the DESCRIPTION.
- 33712 A note indicating that this function need not be reentrant is added to the DESCRIPTION.
- 33713 The %R conversion specification is changed to follow historical practice.

Issue 6

- 33714 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since
- 33715 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*
- 33716 functions.
- 33717
- 33718 The description of %S is updated so that the valid range is [00,60] rather than [00,61].
- 33719 The DESCRIPTION is updated to refer to conversion specifications instead of field descriptors
- 33720 for consistency with other functions.

Issue 7

- 33721 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 33722
- 33723 The description of the *getdate_err* value is expanded.

System Interfaces

getdelim()

NAME

getdelim, getline — read a delimited record from *stream*

SYNOPSIS

```
#include <stdio.h>

ssize_t getdelim(char **restrict lineptr, size_t *restrict n,
    int delimiter, FILE *restrict stream);
ssize_t getline(char **restrict lineptr, size_t *restrict n,
    FILE *restrict stream);
```

DESCRIPTION

The `getdelim()` function shall read from *stream* until it encounters a character matching the *delimiter* character. The *delimiter* argument is an `int`, the value of which the application shall ensure is a character representable as an **unsigned char** of equal value that terminates the read process. If the *delimiter* argument has any other value, the behavior is undefined.

The application shall ensure that **lineptr* is a valid argument that could be passed to the `free()` function. If **n* is non-zero, the application shall ensure that **lineptr* either points to an object of size at least **n* bytes, or is a null pointer.

The size of the object pointed to by **lineptr* shall be increased to fit the incoming line, if it isn't already large enough, including room for the delimiter and a terminating NUL. The characters read, including any delimiter, shall be stored in the string pointed to by the *lineptr* argument, and a terminating NUL added when the delimiter or end of file is encountered.

The `getline()` function shall be equivalent to the `getdelim()` function with the *delimiter* character equal to the `<newline>` character.

The `getdelim()` and `getline()` functions may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of `fgetc()`, `fgets()`, `fread()`, `fscanf()`, `getc()`, `getchar()`, `getdelim()`, `getline()`, `gets()`, or `scanf()` using *stream* that returns data not supplied by a prior call to `ungetc()`.

RETURN VALUE

Upon successful completion, the `getline()` and `getdelim()` functions shall return the number of characters written into the buffer, including the delimiter character if one was encountered before EOF, but excluding the terminating NUL character. If no characters were read, and the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and the function shall return `-1`. If an error occurs, the error indicator for the stream shall be set, and the function shall return `-1` and set `errno` to indicate the error.

ERRORS

For the conditions under which the `getdelim()` and `getline()` functions shall fail and may fail, refer to `fgetc()`.

In addition, these functions shall fail if:

[EINVAL] *lineptr* or *n* is a null pointer.

[ENOMEM] Insufficient memory is available.

These functions may fail if:

[EOVERFLOW] More than `{SSIZE_MAX}` characters were read without encountering the *delimiter* character.

getdelim()*System Interfaces***EXAMPLES**

```

33766
33767     #include <stdio.h>
33768     #include <stdlib.h>
33769
33770     int main(void)
33771     {
33772         FILE *fp;
33773         char *line = NULL;
33774         size_t len = 0;
33775         ssize_t read;
33776         fp = fopen("/etc/motd", "r");
33777         if (fp == NULL)
33778             exit(1);
33779         while ((read = getline(&line, &len, fp)) != -1) {
33780             printf("Retrieved line of length %zu : \n", read);
33781             printf("%s", line);
33782         }
33783         if (ferror(fp)) {
33784             /* handle error */
33785         }
33786         free(line);
33787         fclose(fp);
33788         return 0;
33789     }

```

APPLICATION USAGE

Setting **lineptr* to a null pointer and **n* to zero are allowed and a recommended way to start parsing a file.

The *ferror()* or *feof()* functions should be used to distinguish between an error condition and an end-of-file condition.

Although a NUL terminator is always supplied after the line, note that *strlen(*lineptr)* will be smaller than the return value if the line contains embedded NUL characters.

RATIONALE

These functions are widely used to solve the problem that the *fgets()* function has with long lines. The functions automatically enlarge the target buffers if needed. These are especially useful since they reduce code needed for applications.

FUTURE DIRECTIONS

None.

SEE ALSO

fgetc(), *fgets()*, *free()*

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 7.

33807 NAME

33808 getegid — get the effective group ID

33809 SYNOPSIS

33810 #include <unistd.h>

33811 gid_t getegid(void);

33812 DESCRIPTION

33813 The *getegid()* function shall return the effective group ID of the calling process.

33814 RETURN VALUE

33815 The *getegid()* function shall always be successful and no return value is reserved to indicate an error.

33817 ERRORS

33818 No errors are defined.

33819 EXAMPLES

33820 None.

33821 APPLICATION USAGE

33822 None.

33823 RATIONALE

33824 None.

33825 FUTURE DIRECTIONS

33826 None.

33827 SEE ALSO

33828 *geteuid(), getgid(), getuid(), setegid(), seteuid(), setgid(), setregid(), setreuid(), setuid()*

33829 XBD <sys/types.h>, <unistd.h>

33830 CHANGE HISTORY

33831 First released in Issue 1. Derived from Issue 1 of the SVID.

33832 Issue 6

33833 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

33834 The following new requirements on POSIX implementations derive from alignment with the
33835 Single UNIX Specification:

- 33836 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
33837 required for conforming implementations of previous POSIX specifications, it was not
33838 required for UNIX applications.

getenv()*System Interfaces*33839 **NAME**

33840 getenv — get value of an environment variable

33841 **SYNOPSIS**

33842 #include <stdlib.h>

33843 char *getenv(const char *name);

33844 **DESCRIPTION**

33845 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 33846 conflict between the requirements described here and the ISO C standard is unintentional. This
 33847 volume of POSIX.1-2008 defers to the ISO C standard.

33848 The *getenv()* function shall search the environment of the calling process (see XBD Chapter 8, on
 33849 page 173) for the environment variable *name* if it exists and return a pointer to the value of the
 33850 environment variable. If the specified environment variable cannot be found, a null pointer shall
 33851 be returned. The application shall ensure that it does not modify the string pointed to by the
 33852 *getenv()* function.

33853 CX The string pointed to may be overwritten by a subsequent call to *getenv()*, *setenv()*, *unsetenv()*,
 33854 XSI or *putenv()* but shall not be overwritten by a call to any other function in this volume of
 33855 POSIX.1-2008.

33856 CX If the application modifies *environ* or the pointers to which it points, the behavior of *getenv()* is
 33857 undefined.

33858 The *getenv()* function need not be thread-safe.

33859 **RETURN VALUE**

33860 Upon successful completion, *getenv()* shall return a pointer to a string containing the *value* for
 33861 the specified *name*. If the specified *name* cannot be found in the environment of the calling
 33862 process, a null pointer shall be returned.

33863 **ERRORS**

33864 No errors are defined.

33865 **EXAMPLES**33866 **Getting the Value of an Environment Variable**

33867 The following example gets the value of the *HOME* environment variable.

```
33868       #include <stdlib.h>
33869       ...
33870       const char *name = "HOME";
33871       char *value;
33872       value = getenv(name);
```

33873 **APPLICATION USAGE**

33874 None.

33875 **RATIONALE**

33876 The *clearenv()* function was considered but rejected. The *putenv()* function has now been
 33877 included for alignment with the Single UNIX Specification.

33878 The *getenv()* function is inherently not thread-safe because it returns a value pointing to static
 33879 data.

33880 Conforming applications are required not to modify *environ* directly, but to use only the
 33881 functions described here to manipulate the process environment as an abstract object. Thus, the

implementation of the environment access functions has complete control over the data structure used to represent the environment (subject to the requirement that *environ* be maintained as a list of strings with embedded <equals-sign> characters for applications that wish to scan the environment). This constraint allows the implementation to properly manage the memory it allocates, either by using allocated storage for all variables (copying them on the first invocation of *setenv()* or *unsetenv()*), or keeping track of which strings are currently in allocated space and which are not, via a separate table or some other means. This enables the implementation to free any allocated space used by strings (and perhaps the pointers to them) stored in *environ* when *unsetenv()* is called. A C runtime start-up procedure (that which invokes *main()* and perhaps initializes *environ*) can also initialize a flag indicating that none of the environment has yet been copied to allocated storage, or that the separate table has not yet been initialized.

In fact, for higher performance of *getenv()*, the implementation could also maintain a separate copy of the environment in a data structure that could be searched much more quickly (such as an indexed hash table, or a binary tree), and update both it and the linear list at *environ* when *setenv()* or *unsetenv()* is invoked.

Performance of *getenv()* can be important for applications which have large numbers of environment variables. Typically, applications like this use the environment as a resource database of user-configurable parameters. The fact that these variables are in the user's shell environment usually means that any other program that uses environment variables (such as *ls*, which attempts to use *COLUMNS*), or really almost any utility (*LANG*, *LC_ALL*, and so on) is similarly slowed down by the linear search through the variables.

An implementation that maintains separate data structures, or even one that manages the memory it consumes, is not currently required as it was thought it would reduce consensus among implementors who do not want to change their historical implementations.

FUTURE DIRECTIONS

A future version may add one or more functions to access and modify the environment in a thread-safe manner.

SEE ALSO

exec, *putenv()*, *setenv()*, *unsetenv()*

XBD Chapter 8 (on page 173), <stdlib.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 6

The following changes were made to align with the IEEE P1003.1a draft standard:

- References added to the new *setenv()* and *unsetenv()* functions.

The normative text is updated to avoid use of the term “must” for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #062 is applied, clarifying that a call to *putenv()* may also cause the string to be overwritten.

Austin Group Interpretation 1003.1-2001 #148 is applied, adding the FUTURE DIRECTIONS.

getenv()

System Interfaces

33927

Austin Group Interpretation 1003.1-2001 #156 is applied.

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

System Interfaces

geteuid()33928 **NAME**

33929 geteuid — get the effective user ID

33930 **SYNOPSIS**

33931 #include <unistd.h>

33932 uid_t geteuid(void);

33933 **DESCRIPTION**33934 The *geteuid()* function shall return the effective user ID of the calling process.33935 **RETURN VALUE**33936 The *geteuid()* function shall always be successful and no return value is reserved to indicate an
33937 error.33938 **ERRORS**

33939 No errors are defined.

33940 **EXAMPLES**

33941 None.

33942 **APPLICATION USAGE**

33943 None.

33944 **RATIONALE**

33945 None.

33946 **FUTURE DIRECTIONS**

33947 None.

33948 **SEE ALSO**33949 *getegid(), getgid(), getuid(), setegid(), seteuid(), setgid(), setregid(), setreuid(), setuid()*

33950 XBD <sys/types.h>, <unistd.h>

33951 **CHANGE HISTORY**

33952 First released in Issue 1. Derived from Issue 1 of the SVID.

33953 **Issue 6**

33954 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

33955 The following new requirements on POSIX implementations derive from alignment with the
33956 Single UNIX Specification:

- 33957 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
33958 required for conforming implementations of previous POSIX specifications, it was not
33959 required for UNIX applications.

getgid()*System Interfaces***33960 NAME**

33961 getgid — get the real group ID

33962 SYNOPSIS

33963 #include <unistd.h>

33964 gid_t getgid(void);

33965 DESCRIPTION33966 The *getgid()* function shall return the real group ID of the calling process.**33967 RETURN VALUE**33968 The *getgid()* function shall always be successful and no return value is reserved to indicate an error.
33969**33970 ERRORS**

33971 No errors are defined.

33972 EXAMPLES

33973 None.

33974 APPLICATION USAGE

33975 None.

33976 RATIONALE

33977 None.

33978 FUTURE DIRECTIONS

33979 None.

33980 SEE ALSO33981 *getegid(), geteuid(), getuid(), setegid(), seteuid(), setgid(), setregid(), setreuid(), setuid()*

33982 XBD <sys/types.h>, <unistd.h>

33983 CHANGE HISTORY

33984 First released in Issue 1. Derived from Issue 1 of the SVID.

33985 Issue 6

33986 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

33987 The following new requirements on POSIX implementations derive from alignment with the
33988 Single UNIX Specification:

- 33989 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
33990 required for conforming implementations of previous POSIX specifications, it was not
33991 required for UNIX applications.

*System Interfaces***getgrent()**33992 **NAME**

33993 getgrent — get the group database entry

33994 **SYNOPSIS**

33995 XSI #include <grp.h>

33996 struct group *getgrent(void);

33997 **DESCRIPTION**33998 Refer to *endgrent()*.

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

getgrgid()*System Interfaces*33999 **NAME**34000 **getgrgid, getgrgid_r** — get group database entry for a group ID34001 **SYNOPSIS**

```
34002 #include <grp.h>
34003
34003 struct group *getgrgid(gid_t gid);
34004 int getgrgid_r(gid_t gid, struct group *grp, char *buffer,
34005               size_t bufsize, struct group **result);
```

34006 **DESCRIPTION**34007 The *getgrgid()* function shall search the group database for an entry with a matching *gid*.34008 The *getgrgid()* function need not be thread-safe.

34009 The *getgrgid_r()* function shall update the **group** structure pointed to by *grp* and store a pointer to that structure at the location pointed to by *result*. The structure shall contain an entry from the group database with a matching *gid*. Storage referenced by the group structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to *sysconf(_SC_GETGR_R_SIZE_MAX)* returns either -1 without changing *errno* or an initial value suggested for the size of this buffer. A null pointer shall be returned at the location pointed to by *result* on error or if the requested entry is not found.

34016 **RETURN VALUE**

34017 Upon successful completion, *getgrgid()* shall return a pointer to a **struct group** with the structure defined in **<grp.h>** with a matching entry if one is found. The *getgrgid()* function shall return a null pointer if either the requested entry was not found, or an error occurred. On error, *errno* shall be set to indicate the error.

34021 The return value may point to a static area which is overwritten by a subsequent call to *getgrent()*, *getgrgid()*, or *getgrnam()*.

34023 If successful, the *getgrgid_r()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

34025 **ERRORS**34026 The *getgrgid()* and *getgrgid_r()* functions may fail if:

34027 [EIO] An I/O error has occurred.

34028 [EINTR] A signal was caught during *getgrgid()*.

34029 [EMFILE] All file descriptors available to the process are currently open.

34030 [ENFILE] The maximum allowable number of files is currently open in the system.

34031 The *getgrgid_r()* function may fail if:

34032 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting **group** structure.

EXAMPLES

Note that *sysconf*(*_SC_GETGR_R_SIZE_MAX*) may return -1 if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with *getgrgid_r*().

```

long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct group result;
struct group *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getgrgid_r(42, &result, buffer, len, &resultp)) == ERANGE)
{
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
}
if (e != 0)
    ...handle error...
free (buffer);

```

Finding an Entry in the Group Database

The following example uses *getgrgid*() to search the group database for a group ID that was previously stored in a *stat* structure, then prints out the group name if it is found. If the group is not found, the program prints the numeric value of the group for the entry.

```

#include <sys/types.h>
#include <grp.h>
#include <stdio.h>
...
struct stat statbuf;
struct group *grp;
...
if ((grp = getgrgid(statbuf.st_gid)) != NULL)
    printf(" %-8.8s", grp->gr_name);
else
    printf(" %-8d", statbuf.st_gid);
...

```

getgrgid()*System Interfaces***APPLICATION USAGE**

Applications wishing to check for error situations should set *errno* to 0 before calling *getgrgid()*. If *errno* is set on return, an error occurred.

The *getgrgid_r()* function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return -1 from *sysconf()* indicating that there is no maximum for *_SC_GETGR_R_SIZE_MAX*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endgrent(), *getgrnam()*, *sysconf()*

XBD *<grp.h>*, *<sys/types.h>*

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getgrgid_r()* function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getgrgid()* function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getgrgid_r()* function is marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *gid*.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

In the SYNOPSIS, the optional include of the *<sys/types.h>* header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

34120 **Issue 7**

34121 Austin Group Interpretation 1003.1-2001 #156 is applied.

34122 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

34123 SD5-XSH-ERN-166 is applied.

34124 The *getgrgid_r()* function is moved from the Thread-Safe Functions option to the Base.

34125 A minor addition is made to the EXAMPLES section, reminding the application developer to
34126 free memory allocated as if by *malloc()*.

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

getgrnam()*System Interfaces*34127 **NAME**34128 `getgrnam, getgrnam_r` — search group database for a name34129 **SYNOPSIS**

```

34130 #include <grp.h>
34131 struct group *getgrnam(const char *name);
34132 int getgrnam_r(const char *name, struct group *grp, char *buffer,
34133               size_t bufsize, struct group **result);

```

34134 **DESCRIPTION**34135 The `getgrnam()` function shall search the group database for an entry with a matching *name*.34136 The `getgrnam()` function need not be thread-safe.

34137 The `getgrnam_r()` function shall update the **group** structure pointed to by *grp* and store a pointer
 34138 to that structure at the location pointed to by *result*. The structure shall contain an entry from
 34139 the group database with a matching *name*. Storage referenced by the **group** structure is allocated
 34140 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to
 34141 `sysconf(_SC_GETGR_R_SIZE_MAX)` returns either `-1` without changing *errno* or an initial value
 34142 suggested for the size of this buffer. A null pointer is returned at the location pointed to by *result*
 34143 on error or if the requested entry is not found.

34144 **RETURN VALUE**

34145 The `getgrnam()` function shall return a pointer to a **struct group** with the structure defined in
 34146 `<grp.h>` with a matching entry if one is found. The `getgrnam()` function shall return a null
 34147 pointer if either the requested entry was not found, or an error occurred. On error, *errno* shall be
 34148 set to indicate the error.

34149 The return value may point to a static area which is overwritten by a subsequent call to
 34150 `getgrent()`, `getgrgid()`, or `getgrnam()`.

34151 The `getgrnam_r()` function shall return zero on success or if the requested entry was not found
 34152 and no error has occurred. If any error has occurred, an error number shall be returned to
 34153 indicate the error.

34154 **ERRORS**34155 The `getgrnam()` and `getgrnam_r()` functions may fail if:

34156 [EIO] An I/O error has occurred.

34157 [EINTR] A signal was caught during `getgrnam()`.

34158 [EMFILE] All file descriptors available to the process are currently open.

34159 [ENFILE] The maximum allowable number of files is currently open in the system.

34160 The `getgrnam_r()` function may fail if:

34161 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be
 34162 referenced by the resulting **group** structure.

EXAMPLES

Note that `sysconf(_SC_GETGR_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with `getgrnam_r()`.

```

long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct group result;
struct group *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getgrnam_r("somegroup", &result, buffer, len, &resultp))
        == ERANGE)
{
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
}
if (e != 0)
    ...handle error...
free (buffer);

```

APPLICATION USAGE

Applications wishing to check for error situations should set `errno` to 0 before calling `getgrnam()`. If `errno` is set on return, an error occurred.

The `getgrnam_r()` function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return `-1` from `sysconf()` indicating that there is no maximum for `_SC_GETGR_R_SIZE_MAX`.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`endgrent()`, `getgrgid()`, `sysconf()`

XBD `<grp.h>`, `<sys/types.h>`

getgrnam()*System Interfaces***CHANGE HISTORY**

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getgrnam_r()* function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getgrnam()* function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getgrnam_r()* function is marked as part of the Thread-Safe Functions option.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

Issue 7

Austin Group Interpretation 1003.1-2001 #081 is applied, clarifying the RETURN VALUE section.

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied.

The *getgrnam_r()* function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by *malloc()*.

NAME

getgroups — get supplementary group IDs

SYNOPSIS

```
#include <unistd.h>
```

```
int getgroups(int gidsetsize, gid_t grouplist[]);
```

DESCRIPTION

The *getgroups()* function shall fill in the array *grouplist* with the current supplementary group IDs of the calling process. It is implementation-defined whether *getgroups()* also returns the effective group ID in the *grouplist* array.

The *gidsetsize* argument specifies the number of elements in the array *grouplist*. The actual number of group IDs stored in the array shall be returned. The values of array entries with indices greater than or equal to the value returned are undefined.

If *gidsetsize* is 0, *getgroups()* shall return the number of group IDs that it would otherwise return without modifying the array pointed to by *grouplist*.

If the effective group ID of the process is returned with the supplementary group IDs, the value returned shall always be greater than or equal to one and less than or equal to the value of {NGROUPS_MAX}+1.

RETURN VALUE

Upon successful completion, the number of supplementary group IDs shall be returned. A return value of -1 indicates failure and *errno* shall be set to indicate the error.

ERRORS

The *getgroups()* function shall fail if:

[EINVAL]	The <i>gidsetsize</i> argument is non-zero and less than the number of group IDs that would have been returned.

EXAMPLES**Getting the Supplementary Group IDs of the Calling Process**

The following example places the current supplementary group IDs of the calling process into the *group* array.

```
#include <sys/types.h>
#include <unistd.h>
...
gid_t *group;
int ngroups;
long ngroups_max;

ngroups_max = sysconf(_SC_NGROUPS_MAX) + 1;
group = (gid_t *)malloc(ngroups_max * sizeof(gid_t));

ngroups = getgroups(ngroups_max, group);
```

APPLICATION USAGE

None.

RATIONALE

The related function *setgroups()* is a privileged operation and therefore is not covered by this volume of POSIX.1-2008.

As implied by the definition of supplementary groups, the effective group ID may appear in the

getgroups()*System Interfaces*

array returned by *getgroups()* or it may be returned only by *getegid()*. Duplication may exist, but the application needs to call *getegid()* to be sure of getting all of the information. Various implementation variations and administrative sequences cause the set of groups appearing in the result of *getgroups()* to vary in order and as to whether the effective group ID is included, even when the set of groups is the same (in the mathematical sense of “set”). (The history of a process and its parents could affect the details of the result.)

Application developers should note that {NGROUPS_MAX} is not necessarily a constant on all implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid(), *setgid()*

XBD *<sys/types.h>*, *<unistd.h>*

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

Issue 6

In the SYNOPSIS, the optional include of the *<sys/types.h>* header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- A return value of 0 is not permitted, because {NGROUPS_MAX} cannot be 0. This is a FIPS requirement.

The following changes were made to align with the IEEE P1003.1a draft standard:

- An explanation is added that the effective group ID may be included in the supplementary group list.

*System Interfaces***gethostent()**34313 **NAME**

34314 gethostent — network host database functions

34315 **SYNOPSIS**

34316 #include <netdb.h>

34317 struct hostent *gethostent(void);

34318 **DESCRIPTION**34319 Refer to *endhostent()*.

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

gethostid()

System Interfaces

34320 **NAME**

34321 gethostid — get an identifier for the current host

34322 **SYNOPSIS**

```
34323 XSI       #include <unistd.h>  
34324       long gethostid(void);
```

34325 **DESCRIPTION**34326 The *gethostid()* function shall retrieve a 32-bit identifier for the current host.34327 **RETURN VALUE**34328 Upon successful completion, *gethostid()* shall return an identifier for the current host.34329 **ERRORS**

34330 No errors are defined.

34331 **EXAMPLES**

34332 None.

34333 **APPLICATION USAGE**

34334 This volume of POSIX.1-2008 does not define the domain in which the return value is unique.

34335 **RATIONALE**

34336 None.

34337 **FUTURE DIRECTIONS**

34338 None.

34339 **SEE ALSO**34340 *initstate()*34341 XBD *<unistd.h>*34342 **CHANGE HISTORY**

34343 First released in Issue 4, Version 2.

34344 **Issue 5**

34345 Moved from X/OPEN UNIX extension to BASE.

NAME

gethostname — get name of current host

SYNOPSIS

```
#include <unistd.h>
```

```
int gethostname(char *name, size_t namelen);
```

DESCRIPTION

The *gethostname()* function shall return the standard host name for the current machine. The *namelen* argument shall specify the size of the array pointed to by the *name* argument. The returned name shall be null-terminated, except that if *namelen* is an insufficient length to hold the host name, then the returned name shall be truncated and it is unspecified whether the returned name is null-terminated.

Host names are limited to {HOST_NAME_MAX} bytes.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

gethostid(), *uname()*

XBD <unistd.h>

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter from *socklen_t* to *size_t*.

getitimer()*System Interfaces*34377 **NAME**34378 **getitimer, setitimer** — get and set value of interval timer34379 **SYNOPSIS**

```

34380 OB XSI #include <sys/time.h>
34381
34381 int getitimer(int which, struct itimerval *value);
34382 int setitimer(int which, const struct itimerval *restrict value,
34383               struct itimerval *restrict ovalue);

```

34384 **DESCRIPTION**

34385 The *getitimer()* function shall store the current value of the timer specified by *which* into the
 34386 structure pointed to by *value*. The *setitimer()* function shall set the timer specified by *which* to the
 34387 value specified in the structure pointed to by *value*, and if *ovalue* is not a null pointer, store the
 34388 previous value of the timer in the structure pointed to by *ovalue*.

34389 A timer value is defined by the **itimerval** structure, specified in **<sys/time.h>**. If *it_value* is non-
 34390 zero, it shall indicate the time to the next timer expiration. If *it_interval* is non-zero, it shall
 34391 specify a value to be used in reloading *it_value* when the timer expires. Setting *it_value* to 0 shall
 34392 disable a timer, regardless of the value of *it_interval*. Setting *it_interval* to 0 shall disable a timer
 34393 after its next expiration (assuming *it_value* is non-zero).

34394 Implementations may place limitations on the granularity of timer values. For each interval
 34395 timer, if the requested timer value requires a finer granularity than the implementation supports,
 34396 the actual timer value shall be rounded up to the next supported value.

34397 An XSI-conforming implementation provides each process with at least three interval timers,
 34398 which are indicated by the *which* argument:

34399	ITIMER_PROF	Decrements both in process virtual time and when the system is running
34400		on behalf of the process. It is designed to be used by interpreters in
34401		statistically profiling the execution of interpreted programs. Each time the
34402		ITIMER_PROF timer expires, the SIGPROF signal is delivered.
34403	ITIMER_REAL	Decrements in real time. A SIGALRM signal is delivered when this timer
34404		expires.
34405	ITIMER_VIRTUAL	Decrements in process virtual time. It runs only when the process is
34406		executing. A SIGVTALRM signal is delivered when it expires.

34407 The interaction between *setitimer()* and *alarm()* or *sleep()* is unspecified.

34408 **RETURN VALUE**

34409 Upon successful completion, *getitimer()* or *setitimer()* shall return 0; otherwise, -1 shall be
 34410 returned and *errno* set to indicate the error.

34411 **ERRORS**

34412 The *setitimer()* function shall fail if:

34413	[EINVAL]	The <i>value</i> argument is not in canonical form. (In canonical form, the number of
34414		microseconds is a non-negative integer less than 1 000 000 and the number of
34415		seconds is a non-negative integer.)

34416 The *getitimer()* and *setitimer()* functions may fail if:

34417	[EINVAL]	The <i>which</i> argument is not recognized.

34418 **EXAMPLES**

34419 None.

34420 **APPLICATION USAGE**

34421 Applications should use the *timer_gettime()* and *timer_settime()* functions instead of the
 34422 obsolescent *getitimer()* and *setitimer()* functions, respectively.

34423 **RATIONALE**

34424 None.

34425 **FUTURE DIRECTIONS**34426 The *getitimer()* and *setitimer()* functions may be removed in a future version.34427 **SEE ALSO**34428 *alarm()*, *exec*, *sleep()*, *timer_getoverrun()*34429 XBD [*<signal.h>*](#), [*<sys/time.h>*](#)34430 **CHANGE HISTORY**

34431 First released in Issue 4, Version 2.

34432 **Issue 5**

34433 Moved from X/OPEN UNIX extension to BASE.

34434 **Issue 6**

34435 The **restrict** keyword is added to the *setitimer()* prototype for alignment with the
 34436 ISO/IEC 9899:1999 standard.

34437 **Issue 7**34438 The *getitimer()* and *setitimer()* functions are marked obsolescent.

getline()*System Interfaces*34439 **NAME**34440 getline — read a delimited record from *stream*34441 **SYNOPSIS**

```
34442 CX      #include <stdio.h>
34443          ssize_t getline(char **restrict lineptr, size_t *restrict n,
34444                          FILE *restrict stream);
```

34445 **DESCRIPTION**34446 Refer to *getdelim()*.

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

NAME

getlogin, getlogin_r — get login name

SYNOPSIS

```
#include <unistd.h>

char *getlogin(void);
int getlogin_r(char *name, size_t namesize);
```

DESCRIPTION

The *getlogin()* function shall return a pointer to a string containing the user name associated by the login activity with the controlling terminal of the current process. If *getlogin()* returns a non-null pointer, then that pointer points to the name that the user logged in under, even if there are several login names with the same user ID.

The *getlogin()* function need not be thread-safe.

The *getlogin_r()* function shall put the name associated by the login activity with the controlling terminal of the current process in the character array pointed to by *name*. The array is *namesize* characters long and should have space for the name and the terminating null character. The maximum size of the login name is {LOGIN_NAME_MAX}.

If *getlogin_r()* is successful, *name* points to the name the user used at login, even if there are several login names with the same user ID.

RETURN VALUE

Upon successful completion, *getlogin()* shall return a pointer to the login name or a null pointer if the user's login name cannot be found. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

The return value from *getlogin()* may point to static data whose content is overwritten by each call.

If successful, the *getlogin_r()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EMFILE] All file descriptors available to the process are currently open.

[ENFILE] The maximum allowable number of files is currently open in the system.

[ENXIO] The calling process has no controlling terminal.

The *getlogin_r()* function may fail if:

[ERANGE] The value of *namesize* is smaller than the length of the string to be returned including the terminating null character.

EXAMPLES**Getting the User Login Name**

The following example calls the *getlogin()* function to obtain the name of the user associated with the calling process, and passes this information to the *getpwnam()* function to get the associated user database information.

```
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>
```

getlogin()*System Interfaces*

```

34489     #include <stdio.h>
34490     ...
34491     char *lgn;
34492     struct passwd *pw;
34493     ...
34494     if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
34495         fprintf(stderr, "Get of user information failed.\n"); exit(1);
34496     }

```

APPLICATION USAGE

Three names associated with the current process can be determined: *getpwuid(getuid())* shall return the name associated with the effective user ID of the process; *getlogin()* shall return the name associated with the current login activity; and *getpwuid(getuid())* shall return the name associated with the real user ID of the process.

The *getlogin_r()* function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

The *getlogin()* function returns a pointer to the user's login name. The same user ID may be shared by several login names. If it is desired to get the user database entry that is used during login, the result of *getlogin()* should be used to provide the argument to the *getpwnam()* function. (This might be used to determine the user's login shell, particularly where a single user has multiple login shells with distinct login names, but the same user ID.)

The information provided by the *cuserid()* function, which was originally defined in the POSIX.1-1988 standard and subsequently removed, can be obtained by the following:

```
getpwuid(getuid())
```

while the information provided by historical implementations of *cuserid()* can be obtained by:

```
getpwuid(getuid())
```

The thread-safe version of this function places the user name in a user-supplied buffer and returns a non-zero value if it fails. The non-thread-safe version may return the name in a static data area that may be overwritten by each call.

FUTURE DIRECTIONS

None.

SEE ALSO

getpwnam(), *getpwuid()*, *geteuid()*, *getuid()*

XBD *<limits.h>*, *<unistd.h>*

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getlogin_r()* function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getlogin()* function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getlogin_r()* function is marked as part of the Thread-Safe Functions option.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error:

The *getlogin_r()* function is moved from the Thread-Safe Functions option to the Base.

getmsg()*System Interfaces*34544 **NAME**34545 **getmsg, getpmsg** — receive next message from a STREAMS file (**STREAMS**)34546 **SYNOPSIS**

```

34547 OB XSR #include <stropts.h>
34548
34548 int getmsg(int fildes, struct strbuf *restrict ctlptr,
34549           struct strbuf *restrict dataptr, int *restrict flagsp);
34550
34550 int getpmsg(int fildes, struct strbuf *restrict ctlptr,
34551            struct strbuf *restrict dataptr, int *restrict bandp,
34552            int *restrict flagsp);

```

34553 **DESCRIPTION**

34554 The *getmsg()* function shall retrieve the contents of a message located at the head of the
 34555 STREAM head read queue associated with a STREAMS file and place the contents into one or
 34556 more buffers. The message contains either a data part, a control part, or both. The data and
 34557 control parts of the message shall be placed into separate buffers, as described below. The
 34558 semantics of each part are defined by the originator of the message.

34559 The *getpmsg()* function shall be equivalent to *getmsg()*, except that it provides finer control over
 34560 the priority of the messages received. Except where noted, all requirements on *getmsg()* also
 34561 pertain to *getpmsg()*.

34562 The *fildes* argument specifies a file descriptor referencing a STREAMS-based file.

34563 The *ctlptr* and *dataptr* arguments each point to a **strbuf** structure, in which the *buf* member points
 34564 to a buffer in which the data or control information is to be placed, and the *maxlen* member
 34565 indicates the maximum number of bytes this buffer can hold. On return, the *len* member shall
 34566 contain the number of bytes of data or control information actually received. The *len* member
 34567 shall be set to 0 if there is a zero-length control or data part and *len* shall be set to -1 if no data or
 34568 control information is present in the message.

34569 When *getmsg()* is called, *flagsp* should point to an integer that indicates the type of message the
 34570 process is able to receive. This is described further below.

34571 The *ctlptr* argument is used to hold the control part of the message, and *dataptr* is used to hold
 34572 the data part of the message. If *ctlptr* (or *dataptr*) is a null pointer or the *maxlen* member is -1, the
 34573 control (or data) part of the message shall not be processed and shall be left on the STREAM
 34574 head read queue, and if the *ctlptr* (or *dataptr*) is not a null pointer, *len* shall be set to -1. If the
 34575 *maxlen* member is set to 0 and there is a zero-length control (or data) part, that zero-length part
 34576 shall be removed from the read queue and *len* shall be set to 0. If the *maxlen* member is set to 0
 34577 and there are more than 0 bytes of control (or data) information, that information shall be left on
 34578 the read queue and *len* shall be set to 0. If the *maxlen* member in *ctlptr* (or *dataptr*) is less than the
 34579 control (or data) part of the message, *maxlen* bytes shall be retrieved. In this case, the remainder
 34580 of the message shall be left on the STREAM head read queue and a non-zero return value shall
 34581 be provided.

34582 By default, *getmsg()* shall process the first available message on the STREAM head read queue.
 34583 However, a process may choose to retrieve only high-priority messages by setting the integer
 34584 pointed to by *flagsp* to RS_HIPRI. In this case, *getmsg()* shall only process the next message if it is
 34585 a high-priority message. When the integer pointed to by *flagsp* is 0, any available message shall
 34586 be retrieved. In this case, on return, the integer pointed to by *flagsp* shall be set to RS_HIPRI if a
 34587 high-priority message was retrieved, or 0 otherwise.

34588 For *getpmsg()*, the flags are different. The *flagsp* argument points to a bitmask with the following
 34589 mutually-exclusive flags defined: MSG_HIPRI, MSG_BAND, and MSG_ANY. Like *getmsg()*,

getpmsg() shall process the first available message on the STREAM head read queue. A process may choose to retrieve only high-priority messages by setting the integer pointed to by *flagsp* to MSG_HIPRI and the integer pointed to by *bandp* to 0. In this case, *getpmsg()* shall only process the next message if it is a high-priority message. In a similar manner, a process may choose to retrieve a message from a particular priority band by setting the integer pointed to by *flagsp* to MSG_BAND and the integer pointed to by *bandp* to the priority band of interest. In this case, *getpmsg()* shall only process the next message if it is in a priority band equal to, or greater than, the integer pointed to by *bandp*, or if it is a high-priority message. If a process wants to get the first message off the queue, the integer pointed to by *flagsp* should be set to MSG_ANY and the integer pointed to by *bandp* should be set to 0. On return, if the message retrieved was a high-priority message, the integer pointed to by *flagsp* shall be set to MSG_HIPRI and the integer pointed to by *bandp* shall be set to 0. Otherwise, the integer pointed to by *flagsp* shall be set to MSG_BAND and the integer pointed to by *bandp* shall be set to the priority band of the message.

If O_NONBLOCK is not set, *getmsg()* and *getpmsg()* shall block until a message of the type specified by *flagsp* is available at the front of the STREAM head read queue. If O_NONBLOCK is set and a message of the specified type is not present at the front of the read queue, *getmsg()* and *getpmsg()* shall fail and set *errno* to [EAGAIN].

If a hangup occurs on the STREAM from which messages are retrieved, *getmsg()* and *getpmsg()* shall continue to operate normally, as described above, until the STREAM head read queue is empty. Thereafter, they shall return 0 in the *len* members of *ctlptr* and *dataptr*.

RETURN VALUE

Upon successful completion, *getmsg()* and *getpmsg()* shall return a non-negative value. A value of 0 indicates that a full message was read successfully. A return value of MORECTL indicates that more control information is waiting for retrieval. A return value of MOREDATA indicates that more data is waiting for retrieval. A return value of the bitwise-logical OR of MORECTL and MOREDATA indicates that both types of information remain. Subsequent *getmsg()* and *getpmsg()* calls shall retrieve the remainder of the message. However, if a message of higher priority has come in on the STREAM head read queue, the next call to *getmsg()* or *getpmsg()* shall retrieve that higher-priority message before retrieving the remainder of the previous message.

If the high priority control part of the message is consumed, the message shall be placed back on the queue as a normal message of band 0. Subsequent *getmsg()* and *getpmsg()* calls shall retrieve the remainder of the message. If, however, a priority message arrives or already exists on the STREAM head, the subsequent call to *getmsg()* or *getpmsg()* shall retrieve the higher-priority message before retrieving the remainder of the message that was put back.

Upon failure, *getmsg()* and *getpmsg()* shall return -1 and set *errno* to indicate the error.

ERRORS

The *getmsg()* and *getpmsg()* functions shall fail if:

[EAGAIN]	The O_NONBLOCK flag is set and no messages are available.
[EBADF]	The <i>fildev</i> argument is not a valid file descriptor open for reading.
[EBADMSG]	The queued message to be read is not valid for <i>getmsg()</i> or <i>getpmsg()</i> or a pending file descriptor is at the STREAM head.
[EINTR]	A signal was caught during <i>getmsg()</i> or <i>getpmsg()</i> .
[EINVAL]	An illegal value was specified by <i>flagsp</i> , or the STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.

getmsg()*System Interfaces*

34636 [ENOSTR] A STREAM is not associated with *fildev*.

34637 In addition, *getmsg()* and *getpmsg()* shall fail if the STREAM head had processed an

34638 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of

34639 *getmsg()* or *getpmsg()* but reflects the prior error.

EXAMPLES**Getting Any Message**

34641 In the following example, the value of *fd* is assumed to refer to an open STREAMS file. The call

34642 to *getmsg()* retrieves any available message on the associated STREAM-head read queue,

34643 returning control and data information to the buffers pointed to by *ctrlbuf* and *databuf*,

34644 respectively.

```
34646 #include <stropts.h>
34647 ...
34648 int fd;
34649 char ctrlbuf[128];
34650 char databuf[512];
34651 struct strbuf ctrl;
34652 struct strbuf data;
34653 int flags = 0;
34654 int ret;

34655 ctrl.buf = ctrlbuf;
34656 ctrl.maxlen = sizeof(ctrlbuf);

34657 data.buf = databuf;
34658 data.maxlen = sizeof(databuf);

34659 ret = getmsg (fd, &ctrl, &data, &flags);
```

Getting the First Message off the Queue

34661 In the following example, the call to *getpmsg()* retrieves the first available message on the

34662 associated STREAM-head read queue.

```
34663 #include <stropts.h>
34664 ...
34665 int fd;
34666 char ctrlbuf[128];
34667 char databuf[512];
34668 struct strbuf ctrl;
34669 struct strbuf data;
34670 int band = 0;
34671 int flags = MSG_ANY;
34672 int ret;

34673 ctrl.buf = ctrlbuf;
34674 ctrl.maxlen = sizeof(ctrlbuf);

34675 data.buf = databuf;
34676 data.maxlen = sizeof(databuf);

34677 ret = getpmsg (fd, &ctrl, &data, &band, &flags);
```

34678 **APPLICATION USAGE**

34679 None.

34680 **RATIONALE**

34681 None.

34682 **FUTURE DIRECTIONS**34683 The *getmsg()* and *getpmsg()* functions may be removed in a future version.34684 **SEE ALSO**34685 [Section 2.6](#) (on page 494), *poll()*, *putmsg()*, *read()*, *write()*34686 XBD [<stropts.h>](#)34687 **CHANGE HISTORY**

34688 First released in Issue 4, Version 2.

34689 **Issue 5**

34690 Moved from X/OPEN UNIX extension to BASE.

34691 A paragraph regarding “high-priority control parts of messages” is added to the RETURN
34692 VALUE section.34693 **Issue 6**

34694 This function is marked as part of the XSI STREAMS Option Group.

34695 The **restrict** keyword is added to the *getmsg()* and *getpmsg()* prototypes for alignment with the
34696 ISO/IEC 9899:1999 standard.34697 **Issue 7**34698 The *getmsg()* and *getpmsg()* functions are marked obsolescent.

getnameinfo()*System Interfaces*34699 **NAME**34700 `getnameinfo` — get name information34701 **SYNOPSIS**

```
34702 #include <sys/socket.h>
34703 #include <netdb.h>
```

```
34704 int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
34705                char *restrict node, socklen_t nodelen, char *restrict service,
34706                socklen_t servicelen, int flags);
```

34707 **DESCRIPTION**

34708 The `getnameinfo()` function shall translate a socket address to a node name and service location,
 34709 all of which are defined as in `freeaddrinfo()`.

34710 The *sa* argument points to a socket address structure to be translated.

34711 **IPv6** If the socket address structure contains an IPv4-mapped IPv6 address or an IPv4-compatible
 34712 IPv6 address, the implementation shall extract the embedded IPv4 address and lookup the node
 34713 name for that IPv4 address.

34714 If the address is the IPv6 unspecified address ("::"), a lookup shall not be performed and the
 34715 behavior shall be the same as when the node's name cannot be located.

34716 If the *node* argument is non-NULL and the *nodelen* argument is non-zero, then the *node* argument
 34717 points to a buffer able to contain up to *nodelen* characters that receives the node name as a null-
 34718 terminated string. If the *node* argument is NULL or the *nodelen* argument is zero, the node name
 34719 shall not be returned. If the node's name cannot be located, the numeric form of the address
 34720 contained in the socket address structure pointed to by the *sa* argument is returned instead of its
 34721 name.

34722 If the *service* argument is non-NULL and the *servicelen* argument is non-zero, then the *service*
 34723 argument points to a buffer able to contain up to *servicelen* bytes that receives the service name
 34724 as a null-terminated string. If the *service* argument is NULL or the *servicelen* argument is zero,
 34725 the service name shall not be returned. If the service's name cannot be located, the numeric form
 34726 of the service address (for example, its port number) shall be returned instead of its name.

34727 The *flags* argument is a flag that changes the default actions of the function. By default the fully-
 34728 qualified domain name (FQDN) for the host shall be returned, but:

- 34729 • If the flag bit NI_NOFQDN is set, only the node name portion of the FQDN shall be
 34730 returned for local hosts.
- 34731 • If the flag bit NI_NUMERICHOST is set, the numeric form of the address contained in the
 34732 socket address structure pointed to by the *sa* argument shall be returned instead of its
 34733 name.
- 34734 • If the flag bit NI_NAMEREQD is set, an error shall be returned if the host's name cannot
 34735 be located.
- 34736 • If the flag bit NI_NUMERICSERV is set, the numeric form of the service address shall be
 34737 returned (for example, its port number) instead of its name.
- 34738 • If the flag bit NI_NUMERICSERVICE is set, the numeric form of the scope identifier shall be
 34739 returned (for example, interface index) instead of its name. This flag shall be ignored if the
 34740 *sa* argument is not an IPv6 address.
- 34741 • If the flag bit NI_DGRAM is set, this indicates that the service is a datagram service
 34742 (SOCK_DGRAM). The default behavior shall assume that the service is a stream service
 34743 (SOCK_STREAM).

Notes:

1. The two NI_NUMERICxxx flags are required to support the `-n` flag that many commands provide.
2. The NI_DGRAM flag is required for the few AF_INET and AF_INET6 port numbers (for example, [512,514]) that represent different services for UDP and TCP.

The `getnameinfo()` function shall be thread-safe.

RETURN VALUE

A zero return value for `getnameinfo()` indicates successful completion; a non-zero return value indicates failure. The possible values for the failures are listed in the ERRORS section.

Upon successful completion, `getnameinfo()` shall return the *node* and *service* names, if requested, in the buffers provided. The returned names are always null-terminated strings.

ERRORS

The `getnameinfo()` function shall fail and return the corresponding value if:

[EAI_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

[EAI_BADFLAGS]

The *flags* had an invalid value.

[EAI_FAIL]

A non-recoverable error occurred.

[EAI_FAMILY]

The address family was not recognized or the address length was invalid for the specified family.

[EAI_MEMORY] There was a memory allocation failure.

[EAI_NONAME] The name does not resolve for the supplied parameters.

NI_NAMEREQD is set and the host's name cannot be located, or both *nodename* and *servname* were null.

[EAI_OVERFLOW]

An argument buffer overflowed. The buffer pointed to by the *node* argument or the *service* argument was too small.

[EAI_SYSTEM] A system error occurred. The error code can be found in *errno*.

EXAMPLES

None.

APPLICATION USAGE

If the returned values are to be used as part of any further name resolution (for example, passed to `getaddrinfo()`), applications should provide buffers large enough to store any result possible on the system.

Given the IPv4-mapped IPv6 address `::ffff:1.2.3.4`, the implementation performs a lookup as if the socket address structure contains the IPv4 address `1.2.3.4`.

The IPv6 unspecified address (`:::`) and the IPv6 loopback address (`:::1`) are not IPv4-compatible addresses.

RATIONALE

None.

getnameinfo()*System Interfaces*34783 **FUTURE DIRECTIONS**

34784 None.

34785 **SEE ALSO**34786 *endservent()*, *freeaddrinfo()*, *gai_strerror()*, *inet_ntop()*, *socket()*34787 XBD *<netdb.h>*, *<sys/socket.h>*34788 **CHANGE HISTORY**

34789 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

34790 The **restrict** keyword is added to the *getnameinfo()* prototype for alignment with the
34791 ISO/IEC 9899:1999 standard.34792 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/23 is applied, making various changes in
34793 the SYNOPSIS and DESCRIPTION for alignment with IPv6.34794 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/24 is applied, adding the
34795 [EAL_OVERFLOW] error to the ERRORS section.34796 **Issue 7**34797 SD5-XSH-ERN-127 is applied, clarifying the behavior if the address is the IPv6 unspecified
34798 address.

*System Interfaces***getnetbyaddr()**34799 **NAME**

34800 getnetbyaddr, getnetbyname, getnetent — network database functions

34801 **SYNOPSIS**

34802 #include <netdb.h>

34803 struct netent *getnetbyaddr(uint32_t net, int type);

34804 struct netent *getnetbyname(const char *name);

34805 struct netent *getnetent(void);

34806 **DESCRIPTION**34807 Refer to *endnetent()*.

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

getopt()*System Interfaces*34808 **NAME**34809 `getopt, optarg, opterr, optind, optopt` — command option parsing34810 **SYNOPSIS**

```
34811 #include <unistd.h>
34812 int getopt(int argc, char * const argv[], const char *optstring);
34813 extern char *optarg;
34814 extern int opterr, optind, optopt;
```

34815 **DESCRIPTION**

34816 The `getopt()` function is a command-line parser that shall follow Utility Syntax Guidelines 3, 4, 5,
 34817 6, 7, 9, and 10 in XBD [Section 12.2](#) (on page 215).

34818 The parameters `argc` and `argv` are the argument count and argument array as passed to `main()`
 34819 (see `exec()`). The argument `optstring` is a string of recognized option characters; if a character is
 34820 followed by a <colon>, the option takes an argument. All option characters allowed by Utility
 34821 Syntax Guideline 3 are allowed in `optstring`. The implementation may accept other characters as
 34822 an extension.

34823 The variable `optind` is the index of the next element of the `argv[]` vector to be processed. It shall
 34824 be initialized to 1 by the system, and `getopt()` shall update it when it finishes with each element
 34825 of `argv[]`. When an element of `argv[]` contains multiple option characters, it is unspecified how
 34826 `getopt()` determines which options have already been processed.

34827 The `getopt()` function shall return the next option character (if one is found) from `argv` that
 34828 matches a character in `optstring`, if there is one that matches. If the option takes an argument,
 34829 `getopt()` shall set the variable `optarg` to point to the option-argument as follows:

- 34830 1. If the option was the last character in the string pointed to by an element of `argv`, then
 34831 `optarg` shall contain the next element of `argv`, and `optind` shall be incremented by 2. If the
 34832 resulting value of `optind` is greater than `argc`, this indicates a missing option-argument,
 34833 and `getopt()` shall return an error indication.
- 34834 2. Otherwise, `optarg` shall point to the string following the option character in that element
 34835 of `argv`, and `optind` shall be incremented by 1.

34836 If, when `getopt()` is called:

- 34837 `argv[optind]` is a null pointer
- 34838 `*argv[optind]` is not the character `-`
- 34839 `argv[optind]` points to the string `"-"`

34840 `getopt()` shall return `-1` without changing `optind`. If:

- 34841 `argv[optind]` points to the string `--`

34842 `getopt()` shall return `-1` after incrementing `optind`.

34843 If `getopt()` encounters an option character that is not contained in `optstring`, it shall return the
 34844 <question-mark> (`' ? '`) character. If it detects a missing option-argument, it shall return the
 34845 <colon> character (`' : '`) if the first character of `optstring` was a <colon>, or a <question-mark>
 34846 character (`' ? '`) otherwise. In either case, `getopt()` shall set the variable `optopt` to the option
 34847 character that caused the error. If the application has not set the variable `opterr` to 0 and the first
 34848 character of `optstring` is not a <colon>, `getopt()` shall also print a diagnostic message to `stderr` in
 34849 the format specified for the `getopts` utility.

34850 The `getopt()` function need not be thread-safe.

RETURN VALUE

The *getopt()* function shall return the next option character specified on the command line.

A <colon> (':') shall be returned if *getopt()* detects a missing argument and the first character of *optstring* was a <colon> (':').

A <question-mark> ('?') shall be returned if *getopt()* encounters an option character not in *optstring* or detects a missing argument and the first character of *optstring* was not a <colon> (':').

Otherwise, *getopt()* shall return -1 when all command line options are parsed.

ERRORS

No errors are defined.

EXAMPLES**Parsing Command Line Options**

The following code fragment shows how you might process the arguments for a utility that can take the mutually-exclusive options *a* and *b* and the options *f* and *o*, both of which require arguments:

```
#include <unistd.h>

int
main(int argc, char *argv[ ])
{
    int c;
    int bflg, aflag, errflag;
    char *ifile;
    char *ofile;
    extern char *optarg;
    extern int optind, optopt;
    . . .
    while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
        switch(c) {
            case 'a':
                if (bflg)
                    errflag++;
                else
                    aflag++;
                break;
            case 'b':
                if (aflag)
                    errflag++;
                else {
                    bflg++;
                    bproc();
                }
                break;
            case 'f':
                ifile = optarg;
                break;
            case 'o':
                ofile = optarg;
```

getopt()*System Interfaces*

```

34898         break;
34899         case ':': /* -f or -o without operand */
34900             fprintf(stderr,
34901                 "Option -%c requires an operand\n", optopt);
34902             errflg++;
34903             break;
34904         case '?':
34905             fprintf(stderr,
34906                 "Unrecognized option: -%c\n", optopt);
34907             errflg++;
34908     }
34909 }
34910 if (errflg) {
34911     fprintf(stderr, "usage: . . . ");
34912     exit(2);
34913 }
34914 for ( ; optind < argc; optind++) {
34915     if (access(argv[optind], R_OK)) {
34916         . . .
34917     }

```

This code accepts any of the following as equivalent:

```

34919 cmd -ao arg path path
34920 cmd -a -o arg path path
34921 cmd -o arg -a path path
34922 cmd -a -o arg -- path path
34923 cmd -a -oarg path path
34924 cmd -aoarg path path

```

Checking Options and Arguments

The following example parses a set of command line options and prints messages to standard output for each option and argument that it encounters.

```

34928 #include <unistd.h>
34929 #include <stdio.h>
34930 ...
34931 int c;
34932 char *filename;
34933 extern char *optarg;
34934 extern int optind, optopt, opterr;
34935 ...
34936 while ((c = getopt(argc, argv, ":abf:")) != -1) {
34937     switch(c) {
34938         case 'a':
34939             printf("a is set\n");
34940             break;
34941         case 'b':
34942             printf("b is set\n");
34943             break;
34944         case 'f':
34945             filename = optarg;

```

```

34946         printf("filename is %s\n", filename);
34947         break;
34948     case ':':
34949         printf("-%c without filename\n", optopt);
34950         break;
34951     case '?':
34952         printf("unknown arg %c\n", optopt);
34953         break;
34954     }
34955 }

```

Selecting Options from the Command Line

The following example selects the type of database routines the user wants to use based on the *Options* argument.

```

34959 #include <unistd.h>
34960 #include <string.h>
34961 ...
34962 char *Options = "hdbt1";
34963 ...
34964 int dbtype, i;
34965 char c;
34966 char *st;
34967 ...
34968 dbtype = 0;
34969 while ((c = getopt(argc, argv, Options)) != -1) {
34970     if ((st = strchr(Options, c)) != NULL) {
34971         dbtype = st - Options;
34972         break;
34973     }
34974 }

```

APPLICATION USAGE

The *getopt()* function is only required to support option characters included in Utility Syntax Guideline 3. Many historical implementations of *getopt()* support other characters as options. This is an allowed extension, but applications that use extensions are not maximally portable. Note that support for multi-byte option characters is only possible when such characters can be represented as type **int**.

RATIONALE

The *optopt* variable represents historical practice and allows the application to obtain the identity of the invalid option.

The description has been written to make it clear that *getopt()*, like the *getopts* utility, deals with option-arguments whether separated from the option by <blank> characters or not. Note that the requirements on *getopt()* and *getopts* are more stringent than the Utility Syntax Guidelines.

The *getopt()* function shall return **-1**, rather than EOF, so that **<stdio.h>** is not required.

The special significance of a <colon> as the first character of *optstring* makes *getopt()* consistent with the *getopts* utility. It allows an application to make a distinction between a missing argument and an incorrect option letter without having to examine the option letter. It is true that a missing argument can only be detected in one case, but that is a case that has to be considered.

getopt()*System Interfaces*34993 **FUTURE DIRECTIONS**

34994 None.

34995 **SEE ALSO**34996 *exec*34997 XBD [Section 12.2](#) (on page 215), [<unistd.h>](#)34998 **CHANGE HISTORY**

34999 First released in Issue 1. Derived from Issue 1 of the SVID.

35000 **Issue 5**35001 A note indicating that the *getopt()* function need not be reentrant is added to the DESCRIPTION.35002 **Issue 6**

35003 IEEE PASC Interpretation 1003.2 #150 is applied.

35004 Austin Group Interpretation 1003.1-2001 #156 is applied.

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

NAME

getpeername — get the name of the peer socket

SYNOPSIS

```
#include <sys/socket.h>
```

```
int getpeername(int socket, struct sockaddr *restrict address,
socklen_t *restrict address_len);
```

DESCRIPTION

The *getpeername()* function shall retrieve the peer address of the specified socket, store this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this address in the object pointed to by the *address_len* argument.

If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated.

If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by *address* is unspecified.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *getpeername()* function shall fail if:

[EBADF] The *socket* argument is not a valid file descriptor.

[EINVAL] The socket has been shut down.

[ENOTCONN] The socket is not connected or otherwise has not had the peer pre-specified.

[ENOTSOCK] The *socket* argument does not refer to a socket.

[EOPNOTSUPP] The operation is not supported for the socket protocol.

The *getpeername()* function may fail if:

[ENOBUFS] Insufficient resources were available in the system to complete the call.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

accept(), *bind()*, *getsockname()*, *socket()*

XBD *<sys/socket.h>*

getpeername()*System Interfaces***CHANGE HISTORY**

35042	First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
35043	
35044	The restrict keyword is added to the <i>getpeername()</i> prototype for alignment with the
35045	ISO/IEC 9899:1999 standard.

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

NAME

getpgid — get the process group ID for a process

SYNOPSIS

```
#include <unistd.h>
```

```
pid_t getpgid(pid_t pid);
```

DESCRIPTION

The *getpgid()* function shall return the process group ID of the process whose process ID is equal to *pid*. If *pid* is equal to 0, *getpgid()* shall return the process group ID of the calling process.

RETURN VALUE

Upon successful completion, *getpgid()* shall return a process group ID. Otherwise, it shall return (**pid_t**)−1 and set *errno* to indicate the error.

ERRORS

The *getpgid()* function shall fail if:

[EPERM] The process whose process ID is equal to *pid* is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.

[ESRCH] There is no process with a process ID equal to *pid*.

The *getpgid()* function may fail if:

[EINVAL] The value of the *pid* argument is invalid.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fork()*, *getpgid()*, *getpid()*, *getsid()*, *setpgid()*, *setsid()*

XBD <unistd.h>

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 7

The *getpgid()* function is moved from the XSI option to the Base.

getpgrp()

System Interfaces

35082 **NAME**35083 `getpgrp` — get the process group ID of the calling process35084 **SYNOPSIS**35085 `#include <unistd.h>`35086 `pid_t getpgrp(void);`35087 **DESCRIPTION**35088 The `getpgrp()` function shall return the process group ID of the calling process.35089 **RETURN VALUE**35090 The `getpgrp()` function shall always be successful and no return value is reserved to indicate an
35091 error.35092 **ERRORS**

35093 No errors are defined.

35094 **EXAMPLES**

35095 None.

35096 **APPLICATION USAGE**

35097 None.

35098 **RATIONALE**35099 4.3 BSD provides a `getpgrp()` function that returns the process group ID for a specified process.
35100 Although this function supports job control, all known job control shells always specify the
35101 calling process with this function. Thus, the simpler System V `getpgrp()` suffices, and the added
35102 complexity of the 4.3 BSD `getpgrp()` is provided by the XSI extension `getpgid()`.35103 **FUTURE DIRECTIONS**

35104 None.

35105 **SEE ALSO**35106 `exec`, `fork()`, `getpgid()`, `getpid()`, `getppid()`, `kill()`, `setpgid()`, `setsid()`35107 XBD `<sys/types.h>`, `<unistd.h>`35108 **CHANGE HISTORY**

35109 First released in Issue 1. Derived from Issue 1 of the SVID.

35110 **Issue 6**35111 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.35112 The following new requirements on POSIX implementations derive from alignment with the
35113 Single UNIX Specification:

- 35114
- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
35115 required for conforming implementations of previous POSIX specifications, it was not
35116 required for UNIX applications.

35117 NAME

35118 **getpid** — get the process ID

35119 SYNOPSIS

35120 #include <unistd.h>

35121 pid_t getpid(void);

35122 DESCRIPTION

35123 The *getpid()* function shall return the process ID of the calling process.

35124 RETURN VALUE

35125 The *getpid()* function shall always be successful and no return value is reserved to indicate an
35126 error.

35127 ERRORS

35128 No errors are defined.

35129 EXAMPLES

35130 None.

35131 APPLICATION USAGE

35132 None.

35133 RATIONALE

35134 None.

35135 FUTURE DIRECTIONS

35136 None.

35137 SEE ALSO

35138 *exec*, *fork()*, *getpgid()*, *getppid()*, *kill()*, *mknod()*, *setpgid()*, *setsid()*

35139 XBD <sys/types.h>, <unistd.h>

35140 CHANGE HISTORY

35141 First released in Issue 1. Derived from Issue 1 of the SVID.

35142 Issue 6

35143 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

35144 The following new requirements on POSIX implementations derive from alignment with the
35145 Single UNIX Specification:

- 35146 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
35147 required for conforming implementations of previous POSIX specifications, it was not
35148 required for UNIX applications.

getpmsg()*System Interfaces*35149 **NAME**

35150 getpmsg — receive next message from a STREAMS file

35151 **SYNOPSIS**

```
35152 OB XSI #include <stropts.h>
35153
35153 int getpmsg(int fildes, struct strbuf *restrict ctlptr,
35154             struct strbuf *restrict dataptr, int *restrict bandp,
35155             int *restrict flagsp);
```

35156 **DESCRIPTION**35157 Refer to *getmsg()*.

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

System Interfaces

getppid()**NAME**

getppid — get the parent process ID

SYNOPSIS

```
#include <unistd.h>
```

```
pid_t getppid(void);
```

DESCRIPTION

The *getppid()* function shall return the parent process ID of the calling process.

RETURN VALUE

The *getppid()* function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fork()*, *getpgid()*, *getpgrp()*, *getpid()*, *kill()*, *setpgid()*, *setsid()*

XBD *<sys/types.h>*, *<unistd.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the *<sys/types.h>* header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

getpriority()*System Interfaces*35190 **NAME**35191 `getpriority, setpriority` — get and set the nice value35192 **SYNOPSIS**

```

35193 XSI      #include <sys/resource.h>
35194
35194      int getpriority(int which, id_t who);
35195      int setpriority(int which, id_t who, int value);

```

35196 **DESCRIPTION**

35197 The `getpriority()` function shall obtain the nice value of a process, process group, or user. The
 35198 `setpriority()` function shall set the nice value of a process, process group, or user to
 35199 `value+{NZERO}`.

35200 Target processes are specified by the values of the `which` and `who` arguments. The `which`
 35201 argument may be one of the following values: `PRIO_PROCESS`, `PRIO_PGRP`, or `PRIO_USER`,
 35202 indicating that the `who` argument is to be interpreted as a process ID, a process group ID, or an
 35203 effective user ID, respectively. A 0 value for the `who` argument specifies the current process,
 35204 process group, or user.

35205 The nice value set with `setpriority()` shall be applied to the process. If the process is multi-
 35206 threaded, the nice value shall affect all system scope threads in the process.

35207 If more than one process is specified, `getpriority()` shall return value `{NZERO}` less than the
 35208 lowest nice value pertaining to any of the specified processes, and `setpriority()` shall set the nice
 35209 values of all of the specified processes to `value+{NZERO}`.

35210 The default nice value is `{NZERO}`; lower nice values shall cause more favorable scheduling.
 35211 While the range of valid nice values is `[0,{NZERO}*2-1]`, implementations may enforce more
 35212 restrictive limits. If `value+{NZERO}` is less than the system's lowest supported nice value,
 35213 `setpriority()` shall set the nice value to the lowest supported value; if `value+{NZERO}` is greater
 35214 than the system's highest supported nice value, `setpriority()` shall set the nice value to the
 35215 highest supported value.

35216 Only a process with appropriate privileges can lower its nice value.

35217 PS/TPS Any processes or threads using `SCHED_FIFO` or `SCHED_RR` shall be unaffected by a call to
 35218 `setpriority()`. This is not considered an error. A process which subsequently reverts to
 35219 `SCHED_OTHER` need not have its priority affected by such a `setpriority()` call.

35220 The effect of changing the nice value may vary depending on the process-scheduling algorithm
 35221 in effect.

35222 Since `getpriority()` can return the value `-1` upon successful completion, it is necessary to set `errno`
 35223 to 0 prior to a call to `getpriority()`. If `getpriority()` returns the value `-1`, then `errno` can be checked
 35224 to see if an error occurred or if the value is a legitimate nice value.

35225 **RETURN VALUE**

35226 Upon successful completion, `getpriority()` shall return an integer in the range `-{NZERO}` to
 35227 `{NZERO}-1`. Otherwise, `-1` shall be returned and `errno` set to indicate the error.

35228 Upon successful completion, `setpriority()` shall return 0; otherwise, `-1` shall be returned and `errno`
 35229 set to indicate the error.

ERRORS

The *getpriority()* and *setpriority()* functions shall fail if:

[ESRCH] No process could be located using the *which* and *who* argument values specified.

[EINVAL] The value of the *which* argument was not recognized, or the value of the *who* argument is not a valid process ID, process group ID, or user ID.

In addition, *setpriority()* may fail if:

[EPERM] A process was located, but neither the real nor effective user ID of the executing process match the effective user ID of the process whose nice value is being changed.

[EACCES] A request was made to change the nice value to a lower numeric value and the current process does not have appropriate privileges.

EXAMPLES**Using getpriority()**

The following example returns the current scheduling priority for the process ID returned by the call to *getpid()*.

```
#include <sys/resource.h>
...
int which = PRIO_PROCESS;
id_t pid;
int ret;

pid = getpid();
ret = getpriority(which, pid);
```

Using setpriority()

The following example sets the priority for the current process ID to -20.

```
#include <sys/resource.h>
...
int which = PRIO_PROCESS;
id_t pid;
int priority = -20;
int ret;

pid = getpid();
ret = setpriority(which, pid, priority);
```

APPLICATION USAGE

The *getpriority()* and *setpriority()* functions work with an offset nice value (nice value -{NZERO}). The nice value is in the range [0,2*{NZERO} -1], while the return value for *getpriority()* and the third parameter for *setpriority()* are in the range [-{NZERO},{NZERO} -1].

RATIONALE

None.

getpriority()*System Interfaces*35269 **FUTURE DIRECTIONS**

35270 None.

35271 **SEE ALSO**35272 *nice()*, *sched_get_priority_max()*, *sched_setscheduler()*35273 XBD [<sys/resource.h>](#)35274 **CHANGE HISTORY**

35275 First released in Issue 4, Version 2.

35276 **Issue 5**

35277 Moved from X/OPEN UNIX extension to BASE.

35278 The DESCRIPTION is reworded in terms of the nice value rather than *priority* to avoid confusion
35279 with functionality in the POSIX Realtime Extension.

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

*System Interfaces***getprotobyname()**35280 **NAME**

35281 getprotobyname, getprotobynumber, getprotoent — network protocol database functions

35282 **SYNOPSIS**

35283 #include <netdb.h>

35284 struct protoent *getprotobyname(const char *name);

35285 struct protoent *getprotobynumber(int proto);

35286 struct protoent *getprotoent(void);

35287 **DESCRIPTION**35288 Refer to *endprotoent()*.

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

getpwent()*System Interfaces*35289 **NAME**

35290 getpwent — get user database entry

35291 **SYNOPSIS**

```
35292 xSI #include <pwd.h>  
35293 struct passwd *getpwent(void);
```

35294 **DESCRIPTION**35295 Refer to *endpwent()*.

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

System Interfaces

getpwnam()**NAME**

getpwnam, getpwnam_r — search user database for a name

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwnam(const char *name);
int getpwnam_r(const char *name, struct passwd *pwd, char *buffer,
               size_t bufsize, struct passwd **result);
```

DESCRIPTION

The *getpwnam()* function shall search the user database for an entry with a matching *name*.

The *getpwnam()* function need not be thread-safe.

Applications wishing to check for error situations should set *errno* to 0 before calling *getpwnam()*. If *getpwnam()* returns a null pointer and *errno* is non-zero, an error occurred.

The *getpwnam_r()* function shall update the **passwd** structure pointed to by *pwd* and store a pointer to that structure at the location pointed to by *result*. The structure shall contain an entry from the user database with a matching *name*. Storage referenced by the structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to *sysconf(_SC_GETPW_R_SIZE_MAX)* returns either -1 without changing *errno* or an initial value suggested for the size of this buffer. A null pointer shall be returned at the location pointed to by *result* on error or if the requested entry is not found.

RETURN VALUE

The *getpwnam()* function shall return a pointer to a **struct passwd** with the structure as defined in **<pwd.h>** with a matching entry if found. A null pointer shall be returned if the requested entry is not found, or an error occurs. On error, *errno* shall be set to indicate the error.

The return value may point to a static area which is overwritten by a subsequent call to *getpwent()*, *getpwnam()*, or *getpwuid()*.

The *getpwnam_r()* function shall return zero on success or if the requested entry was not found and no error has occurred. If an error has occurred, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EIO] An I/O error has occurred.

[EINTR] A signal was caught during *getpwnam()*.

[EMFILE] All file descriptors available to the process are currently open.

[ENFILE] The maximum allowable number of files is currently open in the system.

The *getpwnam_r()* function may fail if:

[ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting **passwd** structure.

getpwnam()*System Interfaces***EXAMPLES**

Note that *sysconf*(*_SC_GETPW_R_SIZE_MAX*) may return -1 if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with *getpwnam_r*().

```

35337 long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
35338 size_t len;
35339 if (initlen == -1)
35340     /* Default initial length. */
35341     len = 1024;
35342 else
35343     len = (size_t) initlen;
35344 struct passwd result;
35345 struct passwd *resultp;
35346 char *buffer = malloc(len);
35347 if (buffer == NULL)
35348     ...handle error...
35349 int e;
35350 while ((e = getpwnam_r("someuser", &result, buffer, len, &resultp))
35351        == ERANGE)
35352     {
35353         size_t newlen = 2 * len;
35354         if (newlen < len)
35355             ...handle error...
35356         len = newlen;
35357         char *newbuffer = realloc(buffer, len);
35358         if (newbuffer == NULL)
35359             ...handle error...
35360         buffer = newbuffer;
35361     }
35362 if (e != 0)
35363     ...handle error...
35364 free (buffer);

```

Getting an Entry for the Login Name

The following example uses the *getlogin*() function to return the name of the user who logged in; this information is passed to the *getpwnam*() function to get the user database entry for that user.

```

35368 #include <sys/types.h>
35369 #include <pwd.h>
35370 #include <unistd.h>
35371 #include <stdio.h>
35372 #include <stdlib.h>
35373 ...
35374 char *lgn;
35375 struct passwd *pw;
35376 ...
35377 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
35378     fprintf(stderr, "Get of user information failed.\n"); exit(1);
35379 }
35380 ...

```

APPLICATION USAGE

Three names associated with the current process can be determined: *getpwuid(getuid())* returns the name associated with the effective user ID of the process; *getlogin()* returns the name associated with the current login activity; and *getpwuid(getuid())* returns the name associated with the real user ID of the process.

The *getpwnam_r()* function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return `-1` from *sysconf()* indicating that there is no maximum for `_SC_GETPW_R_SIZE_MAX`.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpwuid(), *sysconf()*

XBD `<pwd.h>`, `<sys/types.h>`

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getpwnam_r()* function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getpwnam()* function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getpwnam_r()* function is marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *name*.

In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

getpwnam()*System Interfaces*35422 **Issue 7**

35423 Austin Group Interpretation 1003.1-2001 #156 is applied.

35424 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

35425 SD5-XSH-ERN-166 is applied.

35426 The *getpwnam_r()* function is moved from the Thread-Safe Functions option to the Base.35427 A minor addition is made to the EXAMPLES section, reminding the application developer to
35428 free memory allocated as if by *malloc()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

getpwuid()

NAME

getpwuid, getpwuid_r — search user database for a user ID

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwuid(uid_t uid);
int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,
               size_t bufsize, struct passwd **result);
```

DESCRIPTION

The `getpwuid()` function shall search the user database for an entry with a matching *uid*.

The `getpwuid()` function need not be thread-safe.

Applications wishing to check for error situations should set *errno* to 0 before calling `getpwuid()`. If `getpwuid()` returns a null pointer and *errno* is set to non-zero, an error occurred.

The `getpwuid_r()` function shall update the **passwd** structure pointed to by *pwd* and store a pointer to that structure at the location pointed to by *result*. The structure shall contain an entry from the user database with a matching *uid*. Storage referenced by the structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to `sysconf(_SC_GETPW_R_SIZE_MAX)` returns either -1 without changing *errno* or an initial value suggested for the size of this buffer. A null pointer shall be returned at the location pointed to by *result* on error or if the requested entry is not found.

RETURN VALUE

The `getpwuid()` function shall return a pointer to a **struct passwd** with the structure as defined in **<pwd.h>** with a matching entry if found. A null pointer shall be returned if the requested entry is not found, or an error occurs. On error, *errno* shall be set to indicate the error.

The return value may point to a static area which is overwritten by a subsequent call to `getpwent()`, `getpwnam()`, or `getpwuid()`.

If successful, the `getpwuid_r()` function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EIO] An I/O error has occurred.

[EINTR] A signal was caught during `getpwuid()`.

[EMFILE] All file descriptors available to the process are currently open.

[ENFILE] The maximum allowable number of files is currently open in the system.

The `getpwuid_r()` function may fail if:

[ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting **passwd** structure.

getpwuid()*System Interfaces***EXAMPLES**

Note that *sysconf*(*_SC_GETPW_R_SIZE_MAX*) may return -1 if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with *getpwuid_r*().

```

long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct passwd result;
struct passwd *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getpwuid_r(42, &result, buffer, len, &resultp)) == ERANGE)
{
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
}
if (e != 0)
    ...handle error...
free (buffer);

```

Getting an Entry for the Root User

The following example gets the user database entry for the user with user ID 0 (root).

```

#include <sys/types.h>
#include <pwd.h>
...
uid_t id = 0;
struct passwd *pwd;
pwd = getpwuid(id);

```

Finding the Name for the Effective User ID

The following example defines *pws* as a pointer to a structure of type **passwd**, which is used to store the structure pointer returned by the call to the *getpwuid()* function. The *geteuid()* function shall return the effective user ID of the calling process; this is used as the search criteria for the *getpwuid()* function. The call to *getpwuid()* shall return a pointer to the structure containing that user ID value.

```
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>
...
struct passwd *pws;
pws = getpwuid(geteuid());
```

Finding an Entry in the User Database

The following example uses *getpwuid()* to search the user database for a user ID that was previously stored in a **stat** structure, then prints out the user name if it is found. If the user is not found, the program prints the numeric value of the user ID for the entry.

```
#include <sys/types.h>
#include <pwd.h>
#include <stdio.h>
...
struct stat statbuf;
struct passwd *pwd;
...
if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
    printf(" %-8.8s", pwd->pw_name);
else
    printf(" %-8d", statbuf.st_uid);
```

APPLICATION USAGE

Three names associated with the current process can be determined: *getpwuid(geteuid())* returns the name associated with the effective user ID of the process; *getlogin()* returns the name associated with the current login activity; and *getpwuid(getuid())* returns the name associated with the real user ID of the process.

The *getpwuid_r()* function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return **-1** from *sysconf()* indicating that there is no maximum for **_SC_GETPW_R_SIZE_MAX**.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpwnam(), *geteuid()*, *getuid()*, *getlogin()*, *sysconf()*

XBD **<pwd.h>**, **<sys/types.h>**

getpwuid()*System Interfaces***CHANGE HISTORY**

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getpwuid_r()* function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getpwuid()* function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getpwuid_r()* function is marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *uid*.

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied.

The *getpwuid_r()* function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by *malloc()*.

NAME

getrlimit, setrlimit — control maximum resource consumption

SYNOPSIS

```

#include <sys/resource.h>

int getrlimit(int resource, struct rlimit *rlp);
int setrlimit(int resource, const struct rlimit *rlp);

```

DESCRIPTION

The *getrlimit()* function shall get, and the *setrlimit()* function shall set, limits on the consumption of a variety of resources.

Each call to either *getrlimit()* or *setrlimit()* identifies a specific resource to be operated upon as well as a resource limit. A resource limit is represented by an **rlimit** structure. The *rlim_cur* member specifies the current or soft limit and the *rlim_max* member specifies the maximum or hard limit. Soft limits may be changed by a process to any value that is less than or equal to the hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both hard and soft limits can be changed in a single call to *setrlimit()* subject to the constraints described above.

The value RLIM_INFINITY, defined in **<sys/resource.h>**, shall be considered to be larger than any other limit value. If a call to *getrlimit()* returns RLIM_INFINITY for a resource, it means the implementation shall not enforce limits on that resource. Specifying RLIM_INFINITY as any resource limit value on a successful call to *setrlimit()* shall inhibit enforcement of that resource limit.

The following resources are defined:

RLIMIT_CORE	This is the maximum size of a core file, in bytes, that may be created by a process. A limit of 0 shall prevent the creation of a core file. If this limit is exceeded, the writing of a core file shall terminate at this size.
RLIMIT_CPU	This is the maximum amount of CPU time, in seconds, used by a process. If this limit is exceeded, SIGXCPU shall be generated for the process. If the process is catching or ignoring SIGXCPU, or all threads belonging to that process are blocking SIGXCPU, the behavior is unspecified.
RLIMIT_DATA	This is the maximum size of a data segment of the process, in bytes. If this limit is exceeded, the <i>malloc()</i> function shall fail with <i>errno</i> set to [ENOMEM].
RLIMIT_FSIZE	This is the maximum size of a file, in bytes, that may be created by a process. If a write or truncate operation would cause this limit to be exceeded, SIGXFSZ shall be generated for the thread. If the thread is blocking, or the process is catching or ignoring SIGXFSZ, continued attempts to increase the size of a file from end-of-file to beyond the limit shall fail with <i>errno</i> set to [EFBIG].
RLIMIT_NOFILE	This is a number one greater than the maximum value that the system may assign to a newly-created descriptor. If this limit is exceeded, functions that allocate a file descriptor shall fail with <i>errno</i> set to [EMFILE]. This limit constrains the number of file descriptors that a process may allocate.

getrlimit()*System Interfaces*

35623	RLIMIT_STACK	This is the maximum size of the initial thread's stack, in bytes. The implementation does not automatically grow the stack beyond this limit.
35624		If this limit is exceeded, SIGSEGV shall be generated for the thread. If the thread is blocking SIGSEGV, or the process is ignoring or catching SIGSEGV and has not made arrangements to use an alternate stack, the disposition of SIGSEGV shall be set to SIG_DFL before it is generated.
35625		
35626		
35627		
35628		
35629	RLIMIT_AS	This is the maximum size of total available memory of the process, in bytes. If this limit is exceeded, the <i>malloc()</i> and <i>mmap()</i> functions shall fail with <i>errno</i> set to [ENOMEM]. In addition, the automatic stack growth fails with the effects outlined above.
35630		
35631		
35632		
35633	When using the <i>getrlimit()</i> function, if a resource limit can be represented correctly in an object of type rlim_t , then its representation is returned; otherwise, if the value of the resource limit is equal to that of the corresponding saved hard limit, the value returned shall be RLIM_SAVED_MAX; otherwise, the value returned shall be RLIM_SAVED_CUR.	
35634		
35635		
35636		
35637	When using the <i>setrlimit()</i> function, if the requested new limit is RLIM_INFINITY, the new limit shall be "no limit"; otherwise, if the requested new limit is RLIM_SAVED_MAX, the new limit shall be the corresponding saved hard limit; otherwise, if the requested new limit is RLIM_SAVED_CUR, the new limit shall be the corresponding saved soft limit; otherwise, the new limit shall be the requested value. In addition, if the corresponding saved limit can be represented correctly in an object of type rlim_t then it shall be overwritten with the new limit.	
35638		
35639		
35640		
35641		
35642		
35643	The result of setting a limit to RLIM_SAVED_MAX or RLIM_SAVED_CUR is unspecified unless a previous call to <i>getrlimit()</i> returned that value as the soft or hard limit for the corresponding resource limit.	
35644		
35645		
35646	The determination of whether a limit can be correctly represented in an object of type rlim_t is implementation-defined. For example, some implementations permit a limit whose value is greater than RLIM_INFINITY and others do not.	
35647		
35648		
35649	The <i>exec</i> family of functions shall cause resource limits to be saved.	
35650	RETURN VALUE	
35651	Upon successful completion, <i>getrlimit()</i> and <i>setrlimit()</i> shall return 0. Otherwise, these functions shall return -1 and set <i>errno</i> to indicate the error.	
35652		
35653	ERRORS	
35654	The <i>getrlimit()</i> and <i>setrlimit()</i> functions shall fail if:	
35655	[EINVAL]	An invalid <i>resource</i> was specified; or in a <i>setrlimit()</i> call, the new <i>rlim_cur</i> exceeds the new <i>rlim_max</i> .
35656		
35657	[EPERM]	The limit specified to <i>setrlimit()</i> would have raised the maximum limit value, and the calling process does not have appropriate privileges.
35658		
35659	The <i>setrlimit()</i> function may fail if:	
35660	[EINVAL]	The limit specified cannot be lowered because current usage is already higher than the limit.
35661		

EXAMPLES

None.

APPLICATION USAGE

If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the value of `{_POSIX_OPEN_MAX}` from `<limits.h>`, unexpected behavior may occur.

If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the highest currently open file descriptor +1, unexpected behavior may occur.

RATIONALE

It should be noted that RLIMIT_STACK applies “at least” to the stack of the initial thread in the process, and not to the sum of all the stacks in the process, as that would be very limiting unless the value is so big as to provide no value at all with a single thread.

FUTURE DIRECTIONS

None.

SEE ALSO

`exec`, `fork()`, `malloc()`, `open()`, `sigaltstack()`, `sysconf()`, `ulimit()`

XBD `<stropts.h>`, `<sys/resource.h>`

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

An APPLICATION USAGE section is added.

Large File Summit extensions are added.

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/25 is applied, changing wording for RLIMIT_NOFILE in the DESCRIPTION related to functions that allocate a file descriptor failing with [EMFILE]. Text is added to the APPLICATION USAGE section noting the consequences of a process attempting to set the hard or soft limit for RLIMIT_NOFILE less than the highest currently open file descriptor +1.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/46 is applied, updating the definition of RLIMIT_STACK in the DESCRIPTION from “the maximum size of a process stack” to “the maximum size of the initial thread’s stack”. Text is added to the RATIONALE section.

getrusage()*System Interfaces*35693 **NAME**35694 **getrusage** — get information about resource utilization35695 **SYNOPSIS**

```

35696 XSI      #include <sys/resource.h>
35697          int getrusage(int who, struct rusage *r_usage);

```

35698 **DESCRIPTION**

35699 The *getrusage()* function shall provide measures of the resources used by the current process or
 35700 its terminated and waited-for child processes. If the value of the *who* argument is
 35701 RUSAGE_SELF, information shall be returned about resources used by the current process. If the
 35702 value of the *who* argument is RUSAGE_CHILDREN, information shall be returned about
 35703 resources used by the terminated and waited-for children of the current process. If the child is
 35704 never waited for (for example, if the parent has SA_NOCLDWAIT set or sets SIGCHLD to
 35705 SIG_IGN), the resource information for the child process is discarded and not included in the
 35706 resource information provided by *getrusage()*.

35707 The *r_usage* argument is a pointer to an object of type **struct rusage** in which the returned
 35708 information is stored.

35709 **RETURN VALUE**

35710 Upon successful completion, *getrusage()* shall return 0; otherwise, -1 shall be returned and *errno*
 35711 set to indicate the error.

35712 **ERRORS**

35713 The *getrusage()* function shall fail if:

35714 [EINVAL] The value of the *who* argument is not valid.

35715 **EXAMPLES**35716 **Using getrusage()**

35717 The following example returns information about the resources used by the current process.

```

35718      #include <sys/resource.h>
35719      ...
35720      int who = RUSAGE_SELF;
35721      struct rusage usage;
35722      int ret;
35723
35723      ret = getrusage(who, &usage);

```

35724 **APPLICATION USAGE**

35725 None.

35726 **RATIONALE**

35727 None.

35728 **FUTURE DIRECTIONS**

35729 None.

35730 **SEE ALSO**

35731 *exit()*, *sigaction()*, *time()*, *times()*, *wait()*

35732 XBD [<sys/resource.h>](#)

35733	CHANGE HISTORY
35734	First released in Issue 4, Version 2.
35735	Issue 5
35736	Moved from X/OPEN UNIX extension to BASE.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

gets()*System Interfaces*35737 **NAME**35738 gets — get a string from a *stdin* stream35739 **SYNOPSIS**

```
35740 OB #include <stdio.h>
35741 char *gets(char *s);
```

35742 **DESCRIPTION**

35743 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 35744 conflict between the requirements described here and the ISO C standard is unintentional. This
 35745 volume of POSIX.1-2008 defers to the ISO C standard.

35746 The *gets()* function shall read bytes from the standard input stream, *stdin*, into the array pointed
 35747 to by *s*, until a <newline> is read or an end-of-file condition is encountered. Any <newline> shall
 35748 be discarded and a null byte shall be placed immediately after the last byte read into the array.

35749 CX The *gets()* function may mark the last data access timestamp of the file associated with *stream* for
 35750 update. The last data access timestamp shall be marked for update by the first successful
 35751 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*, or
 35752 *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

35753 **RETURN VALUE**

35754 Upon successful completion, *gets()* shall return *s*. If the end-of-file indicator for the stream is set,
 35755 or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *gets()*
 35756 shall return a null pointer. If a read error occurs, the error indicator for the stream shall be set,
 35757 CX *gets()* shall return a null pointer, and set *errno* to indicate the error.

35758 **ERRORS**35759 Refer to *fgetc()*.35760 **EXAMPLES**

35761 None.

35762 **APPLICATION USAGE**

35763 Reading a line that overflows the array pointed to by *s* results in undefined behavior. The use of
 35764 *fgets()* is recommended.

35765 Since the user cannot specify the length of the buffer passed to *gets()*, use of this function is
 35766 discouraged. The length of the string read is unlimited. It is possible to overflow this buffer in
 35767 such a way as to cause applications to fail, or possible system security violations.

35768 Applications should use the *fgets()* function instead of the obsolescent *gets()* function.

35769 **RATIONALE**

35770 The standard developers decided to mark the *gets()* function as obsolescent even though it is in
 35771 the ISO C standard due to the possibility of buffer overflow.

35772 **FUTURE DIRECTIONS**

35773 The *gets()* function may be removed in a future version.

35774 **SEE ALSO**35775 *feof()*, *ferror()*, *fgets()*

35776 XBD <stdio.h>

35777 **CHANGE HISTORY**

35778 First released in Issue 1. Derived from Issue 1 of the SVID.

35779 **Issue 6**

35780 Extensions beyond the ISO C standard are marked.

35781 **Issue 7**

35782 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

35783 The *gets()* function is marked obsolescent.

35784 Changes are made related to support for finegrained timestamps.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

getservbyname()*System Interfaces*35785 **NAME**

35786 getservbyname, getservbyport, getservent — network services database functions

35787 **SYNOPSIS**

35788 #include <netdb.h>

35789 struct servent *getservbyname(const char *name, const char *proto);

35790 struct servent *getservbyport(int port, const char *proto);

35791 struct servent *getservent(void);

35792 **DESCRIPTION**35793 Refer to *endservent()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

getsid()**NAME**

getsid — get the process group ID of a session leader

SYNOPSIS

```
#include <unistd.h>
```

```
pid_t getsid(pid_t pid);
```

DESCRIPTION

The *getsid()* function shall obtain the process group ID of the process that is the session leader of the process specified by *pid*. If *pid* is (**pid_t**)0, it specifies the calling process.

RETURN VALUE

Upon successful completion, *getsid()* shall return the process group ID of the session leader of the specified process. Otherwise, it shall return (**pid_t**)–1 and set *errno* to indicate the error.

ERRORS

The *getsid()* function shall fail if:

[EPERM]	The process specified by <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of the session leader of that process from the calling process.
[ESRCH]	There is no process with a process ID equal to <i>pid</i> .

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fork()*, *getpid()*, *getpgid()*, *setpgid()*, *setsid()*

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 7

The *getsid()* function is moved from the XSI option to the Base.

getsockname()*System Interfaces*35828 **NAME**35829 `getsockname` — get the socket name35830 **SYNOPSIS**35831 `#include <sys/socket.h>`

35832 `int getsockname(int socket, struct sockaddr *restrict address,`
 35833 `socklen_t *restrict address_len);`

35834 **DESCRIPTION**

35835 The `getsockname()` function shall retrieve the locally-bound name of the specified socket, store
 35836 this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of
 35837 this address in the object pointed to by the *address_len* argument.

35838 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
 35839 the stored address shall be truncated.

35840 If the socket has not been bound to a local name, the value stored in the object pointed to by
 35841 *address* is unspecified.

35842 **RETURN VALUE**

35843 Upon successful completion, 0 shall be returned, the *address* argument shall point to the address
 35844 of the socket, and the *address_len* argument shall point to the length of the address. Otherwise, -1
 35845 shall be returned and *errno* set to indicate the error.

35846 **ERRORS**35847 The `getsockname()` function shall fail if:35848 [EBADF] The *socket* argument is not a valid file descriptor.35849 [ENOTSOCK] The *socket* argument does not refer to a socket.

35850 [EOPNOTSUPP] The operation is not supported for this socket's protocol.

35851 The `getsockname()` function may fail if:

35852 [EINVAL] The socket has been shut down.

35853 [ENOBUFS] Insufficient resources were available in the system to complete the function.

35854 **EXAMPLES**

35855 None.

35856 **APPLICATION USAGE**

35857 None.

35858 **RATIONALE**

35859 None.

35860 **FUTURE DIRECTIONS**

35861 None.

35862 **SEE ALSO**35863 `accept()`, `bind()`, `getpeername()`, `socket()`35864 XBD `<sys/socket.h>`

CHANGE HISTORY

35865	
35866	First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
35867	The restrict keyword is added to the <i>getsockname()</i> prototype for alignment with the
35868	ISO/IEC 9899:1999 standard.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

getsockopt()*System Interfaces***NAME**

getsockopt — get the socket options

SYNOPSIS

```
#include <sys/socket.h>

int getsockopt(int socket, int level, int option_name,
               void *restrict option_value, socklen_t *restrict option_len);
```

DESCRIPTION

The *getsockopt()* function manipulates options associated with a socket.

The *getsockopt()* function shall retrieve the value for the option specified by the *option_name* argument for the socket specified by the *socket* argument. If the size of the option value is greater than *option_len*, the value stored in the object pointed to by the *option_value* argument shall be silently truncated. Otherwise, the object pointed to by the *option_len* argument shall be modified to indicate the actual length of the value.

The *level* argument specifies the protocol level at which the option resides. To retrieve options at the socket level, specify the *level* argument as SOL_SOCKET. To retrieve options at other levels, supply the appropriate level identifier for the protocol controlling the option. For example, to indicate that an option is interpreted by the TCP (Transmission Control Protocol), set *level* to IPPROTO_TCP as defined in the *<netinet/in.h>* header.

The socket in use may require the process to have appropriate privileges to use the *getsockopt()* function.

The *option_name* argument specifies a single option to be retrieved. It can be one of the socket-level options defined in *<sys/socket.h>* and described in [Section 2.10.16](#) (on page 522).

RETURN VALUE

Upon successful completion, *getsockopt()* shall return 0; otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *getsockopt()* function shall fail if:

- [EBADF] The *socket* argument is not a valid file descriptor.
 - [EINVAL] The specified option is invalid at the specified socket level.
 - [ENOPROTOOPT] The option is not supported by the protocol.
 - [ENOTSOCK] The *socket* argument does not refer to a socket.
- The *getsockopt()* function may fail if:
- [EACCES] The calling process does not have appropriate privileges.
 - [EINVAL] The socket has been shut down.
 - [ENOBUFFS] Insufficient resources are available in the system to complete the function.

35905 **EXAMPLES**

35906 None.

35907 **APPLICATION USAGE**

35908 None.

35909 **RATIONALE**

35910 None.

35911 **FUTURE DIRECTIONS**

35912 None.

35913 **SEE ALSO**35914 [Section 2.10.16](#) (on page 522), [bind\(\)](#), [close\(\)](#), [endprotoent\(\)](#), [setsockopt\(\)](#), [socket\(\)](#)35915 XBD [<sys/socket.h>](#), [<netinet/in.h>](#)35916 **CHANGE HISTORY**

35917 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

35918 The **restrict** keyword is added to the *getsockopt()* prototype for alignment with the
35919 ISO/IEC 9899:1999 standard.35920 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/47 is applied, updating the description of
35921 SO_LINGER in the DESCRIPTION so that it blocks the calling thread rather than the process.35922 **Issue 7**35923 Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options
35924 that is now in [Section 2.10.16](#) (on page 522).

getsubopt()*System Interfaces*35925 **NAME**

35926 getsubopt — parse suboption arguments from a string

35927 **SYNOPSIS**

35928 #include <stdlib.h>

35929 int getsubopt(char ***optionp*, char * const **keylistp*, char ***valuep*);35930 **DESCRIPTION**35931 The *getsubopt()* function shall parse suboption arguments in a flag argument. Such options often
35932 result from the use of *getopt()*.35933 The *getsubopt()* argument *optionp* is a pointer to a pointer to the option argument string. The
35934 suboption arguments shall be separated by <comma> characters and each may consist of either
35935 a single token, or a token-value pair separated by an <equals-sign>.35936 The *keylistp* argument shall be a pointer to a vector of strings. The end of the vector is identified
35937 by a null pointer. Each entry in the vector is one of the possible tokens that might be found in
35938 **optionp*. Since <comma> characters delimit suboption arguments in *optionp*, they should not
35939 appear in any of the strings pointed to by *keylistp*. Similarly, because an <equals-sign> separates
35940 a token from its value, the application should not include an <equals-sign> in any of the strings
35941 pointed to by *keylistp*.35942 The *valuep* argument is the address of a value string pointer.35943 If a <comma> appears in *optionp*, it shall be interpreted as a suboption separator. After <comma>
35944 characters have been processed, if there are one or more <equals-sign> characters in a suboption
35945 string, the first <equals-sign> in any suboption string shall be interpreted as a separator between
35946 a token and a value. Subsequent <equals-sign> characters in a suboption string shall be
35947 interpreted as part of the value.35948 If the string at **optionp* contains only one suboption argument (equivalently, no <comma>
35949 characters), *getsubopt()* shall update **optionp* to point to the null character at the end of the
35950 string. Otherwise, it shall isolate the suboption argument by replacing the <comma> separator
35951 with a null character, and shall update **optionp* to point to the start of the next suboption
35952 argument. If the suboption argument has an associated value (equivalently, contains an <equals-
35953 sign>), *getsubopt()* shall update **valuep* to point to the value's first character. Otherwise, it shall
35954 set **valuep* to a null pointer. The calling application may use this information to determine
35955 whether the presence or absence of a value for the suboption is an error.35956 Additionally, when *getsubopt()* fails to match the suboption argument with a token in the *keylistp*
35957 array, the calling application should decide if this is an error, or if the unrecognized option
35958 should be processed in another way.35959 **RETURN VALUE**35960 The *getsubopt()* function shall return the index of the matched token string, or -1 if no token
35961 strings were matched.35962 **ERRORS**

35963 No errors are defined.

EXAMPLES

```

35964
35965     #include <stdio.h>
35966     #include <stdlib.h>
35967
35968     int do_all;
35969     const char *type;
35970     int read_size;
35971     int write_size;
35972     int read_only;
35973
35974     enum
35975     {
35976         RO_OPTION = 0,
35977         RW_OPTION,
35978         READ_SIZE_OPTION,
35979         WRITE_SIZE_OPTION
35980     };
35981
35982     const char *mount_opts[] =
35983     {
35984         [RO_OPTION] = "ro",
35985         [RW_OPTION] = "rw",
35986         [READ_SIZE_OPTION] = "rsize",
35987         [WRITE_SIZE_OPTION] = "wsize",
35988         NULL
35989     };
35990
35991     int
35992     main(int argc, char *argv[])
35993     {
35994         char *subopts, *value;
35995         int opt;
35996
35997         while ((opt = getopt(argc, argv, "at:o:")) != -1)
35998             switch(opt)
35999             {
36000                 case 'a':
36001                     do_all = 1;
36002                     break;
36003                 case 't':
36004                     type = optarg;
36005                     break;
36006                 case 'o':
36007                     subopts = optarg;
36008                     while (*subopts != '\0')
36009                         switch(getsubopt(&subopts, mount_opts, &value))
36010                         {
36011                             case RO_OPTION:
36012                                 read_only = 1;
36013                                 break;
36014                             case RW_OPTION:
36015                                 read_only = 0;
36016                                 break;
36017                             case READ_SIZE_OPTION:

```

getsubopt()*System Interfaces*

```

36013         if (value == NULL)
36014             abort();
36015         read_size = atoi(value);
36016         break;
36017     case WRITE_SIZE_OPTION:
36018         if (value == NULL)
36019             abort();
36020         write_size = atoi(value);
36021         break;
36022     default:
36023         /* Unknown suboption. */
36024         printf("Unknown suboption '%s'\n", value);
36025         break;
36026     }
36027     break;
36028 default:
36029     abort();
36030 }
36031
36032 /* Do the real work. */
36033 return 0;
36034 }

```

Parsing Suboptions

The following example uses the `getsubopt()` function to parse a *value* argument in the *optarg* external variable returned by a call to `getopt()`.

```

36037 #include <stdlib.h>
36038 ...
36039 char *tokens[] = {"HOME", "PATH", "LOGNAME", (char *) NULL };
36040 char *value;
36041 int opt, index;
36042
36043 while ((opt = getopt(argc, argv, "e:")) != -1) {
36044     switch(opt) {
36045         case 'e':
36046             while ((index = getsubopt(&optarg, tokens, &value)) != -1) {
36047                 switch(index) {
36048                     ...
36049                 }
36050             }
36051             break;
36052         ...
36053     }
36054 }
36055 ...

```

APPLICATION USAGE

None.

36056 **RATIONALE**

36057 None.

36058 **FUTURE DIRECTIONS**

36059 None.

36060 **SEE ALSO**36061 *getopt()*

36062 XBD <stdlib.h>

36063 **CHANGE HISTORY**

36064 First released in Issue 4, Version 2.

36065 **Issue 5**

36066 Moved from X/OPEN UNIX extension to BASE.

36067 **Issue 6**36068 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/26 is applied, correcting an editorial error
36069 in the SYNOPSIS.36070 **Issue 7**36071 The *getsubopt()* function is moved from the XSI option to the Base.

gettimeofday()

System Interfaces

36072 **NAME**

36073 gettimeofday — get the date and time

36074 **SYNOPSIS**

```
36075 OB XSI #include <sys/time.h>
36076 int gettimeofday(struct timeval *restrict tp, void *restrict tzp);
```

36077 **DESCRIPTION**

36078 The *gettimeofday()* function shall obtain the current time, expressed as seconds and microseconds
 36079 since the Epoch, and store it in the **timeval** structure pointed to by *tp*. The resolution of the
 36080 system clock is unspecified.

36081 If *tzp* is not a null pointer, the behavior is unspecified.

36082 **RETURN VALUE**

36083 The *gettimeofday()* function shall return 0 and no value shall be reserved to indicate an error.

36084 **ERRORS**

36085 No errors are defined.

36086 **EXAMPLES**

36087 None.

36088 **APPLICATION USAGE**

36089 Applications should use the *clock_gettime()* function instead of the obsolescent *gettimeofday()*
 36090 function.

36091 **RATIONALE**

36092 None.

36093 **FUTURE DIRECTIONS**

36094 The *gettimeofday()* function may be removed in a future version.

36095 **SEE ALSO**

36096 *clock_getres()*, *ctime()*

36097 XBD *<sys/time.h>*

36098 **CHANGE HISTORY**

36099 First released in Issue 4, Version 2.

36100 **Issue 5**

36101 Moved from X/OPEN UNIX extension to BASE.

36102 **Issue 6**

36103 The **DESCRIPTION** is updated to refer to “seconds since the Epoch” rather than “seconds since
 36104 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*
 36105 functions.

36106 The **restrict** keyword is added to the *gettimeofday()* prototype for alignment with the
 36107 ISO/IEC 9899:1999 standard.

36108 **Issue 7**

36109 The *gettimeofday()* function is marked obsolescent.

NAME

getuid — get a real user ID

SYNOPSIS

```
#include <unistd.h>
```

```
uid_t getuid(void);
```

DESCRIPTION

The *getuid()* function shall return the real user ID of the calling process.

RETURN VALUE

The *getuid()* function shall always be successful and no return value is reserved to indicate the error.

ERRORS

No errors are defined.

EXAMPLES**Setting the Effective User ID to the Real User ID**

The following example sets the effective user ID and the real user ID of the current process to the real user ID of the caller.

```
#include <unistd.h>
#include <sys/types.h>
...
setreuid(getuid(), getuid());
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid(), *geteuid()*, *getgid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*

XBD [<sys/types.h>](#), [<unistd.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

getutxent()*System Interfaces*36149 **NAME**

36150 getutxent, getutxid, getutxline — get user accounting database entries

36151 **SYNOPSIS**

```
36152 XSI #include <utmpx.h>
36153 struct utmpx *getutxent(void);
36154 struct utmpx *getutxid(const struct utmpx *id);
36155 struct utmpx *getutxline(const struct utmpx *line);
```

36156 **DESCRIPTION**36157 Refer to *endutxent()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

getwc()

NAME

getwc — get a wide character from a stream

SYNOPSIS

```
#include <stdio.h>
```

```
#include <wchar.h>
```

```
wint_t getwc(FILE *stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The `getwc()` function shall be equivalent to `fgetwc()`, except that if it is implemented as a macro it may evaluate *stream* more than once, so the argument should never be an expression with side-effects.

RETURN VALUE

Refer to `fgetwc()`.

ERRORS

Refer to `fgetwc()`.

EXAMPLES

None.

APPLICATION USAGE

Since it may be implemented as a macro, `getwc()` may treat incorrectly a *stream* argument with side-effects. In particular, `getwc(*f++)` does not necessarily work as expected. Therefore, use of this function is not recommended; `fgetwc()` should be used instead.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`fgetwc()`

XBD `<stdio.h>`, `<wchar.h>`

CHANGE HISTORY

First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working draft.

Issue 5

The Optional Header (OH) marking is removed from `<stdio.h>`.

getwchar()

System Interfaces

36193 **NAME**36194 getwchar — get a wide character from a *stdin* stream36195 **SYNOPSIS**

36196 #include <wchar.h>

36197 wint_t getwchar(void);

36198 **DESCRIPTION**

36199 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 36200 conflict between the requirements described here and the ISO C standard is unintentional. This
 36201 volume of POSIX.1-2008 defers to the ISO C standard.

36202 The *getwchar()* function shall be equivalent to *getwc(stdin)*.36203 **RETURN VALUE**36204 Refer to *fgetwc()*.36205 **ERRORS**36206 Refer to *fgetwc()*.36207 **EXAMPLES**

36208 None.

36209 **APPLICATION USAGE**

36210 If the **wint_t** value returned by *getwchar()* is stored into a variable of type **wchar_t** and then
 36211 compared against the **wint_t** macro WEOF, the result may be incorrect. Only the **wint_t** type is
 36212 guaranteed to be able to represent any wide character and WEOF.

36213 **RATIONALE**

36214 None.

36215 **FUTURE DIRECTIONS**

36216 None.

36217 **SEE ALSO**36218 *fgetwc()*, *getwc()*

36219 XBD <wchar.h>

36220 **CHANGE HISTORY**

36221 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working
 36222 draft.

NAME

glob, globfree — generate pathnames matching a pattern

SYNOPSIS

```
#include <glob.h>

int glob(const char *restrict pattern, int flags,
         int (*errfunc)(const char *epath, int eerrno),
         glob_t *restrict pglob);
void globfree(glob_t *pglob);
```

DESCRIPTION

The `glob()` function is a pathname generator that shall implement the rules defined in XCU [Section 2.13](#) (on page 2332), with optional support for rule 3 in XCU [Section 2.13.3](#) (on page 2333).

The structure type `glob_t` is defined in `<glob.h>` and includes at least the following members:

Member Type	Member Name	Description
<code>size_t</code>	<code>gl_pathc</code>	Count of paths matched by <i>pattern</i> .
<code>char **</code>	<code>gl_pathv</code>	Pointer to a list of matched pathnames.
<code>size_t</code>	<code>gl_offs</code>	Slots to reserve at the beginning of <code>gl_pathv</code> .

The argument *pattern* is a pointer to a pathname pattern to be expanded. The `glob()` function shall match all accessible pathnames against this pattern and develop a list of all pathnames that match. In order to have access to a pathname, `glob()` requires search permission on every component of a path except the last, and read permission on each directory of any filename component of *pattern* that contains any of the following special characters: '*', '?', and '['.

The `glob()` function shall store the number of matched pathnames into `pglob->gl_pathc` and a pointer to a list of pointers to pathnames into `pglob->gl_pathv`. The pathnames shall be in sort order as defined by the current setting of the `LC_COLLATE` category; see XBD [Section 7.3.2](#) (on page 146). The first pointer after the last pathname shall be a null pointer. If the pattern does not match any pathnames, the returned number of matched paths is set to 0, and the contents of `pglob->gl_pathv` are implementation-defined.

It is the caller's responsibility to create the structure pointed to by *pglob*. The `glob()` function shall allocate other space as needed, including the memory pointed to by `gl_pathv`. The `globfree()` function shall free any space associated with *pglob* from a previous call to `glob()`.

The *flags* argument is used to control the behavior of `glob()`. The value of *flags* is a bitwise-inclusive OR of zero or more of the following constants, which are defined in `<glob.h>`:

<code>GLOB_APPEND</code>	Append pathnames generated to the ones from a previous call to <code>glob()</code> .
<code>GLOB_DOOFFS</code>	Make use of <code>pglob->gl_offs</code> . If this flag is set, <code>pglob->gl_offs</code> is used to specify how many null pointers to add to the beginning of <code>pglob->gl_pathv</code> . In other words, <code>pglob->gl_pathv</code> shall point to <code>pglob->gl_offs</code> null pointers, followed by <code>pglob->gl_pathc</code> pathname pointers, followed by a null pointer.
<code>GLOB_ERR</code>	Cause <code>glob()</code> to return when it encounters a directory that it cannot open or read. Ordinarily, <code>glob()</code> continues to find matches.
<code>GLOB_MARK</code>	Each pathname that is a directory that matches <i>pattern</i> shall have a <code><slash></code> appended.

glob()*System Interfaces*

- 36267 GLOB_NOCHECK Supports rule 3 in XCU [Section 2.13.3](#) (on page 2333). If *pattern* does not
 36268 match any pathname, then *glob()* shall return a list consisting of only
 36269 *pattern*, and the number of matched pathnames is 1.
- 36270 GLOB_NOESCAPE Disable backslash escaping.
- 36271 GLOB_NOSORT Ordinarily, *glob()* sorts the matching pathnames according to the current
 36272 setting of the *LC_COLLATE* category; see XBD [Section 7.3.2](#) (on page 146).
 36273 When this flag is used, the order of pathnames returned is unspecified.
- 36274 The GLOB_APPEND flag can be used to append a new set of pathnames to those found in a
 36275 previous call to *glob()*. The following rules apply to applications when two or more calls to
 36276 *glob()* are made with the same value of *pglob* and without intervening calls to *globfree()*:
- 36277 1. The first such call shall not set GLOB_APPEND. All subsequent calls shall set it.
 - 36278 2. All the calls shall set GLOB_DOOFFS, or all shall not set it.
 - 36279 3. After the second call, *pglob->gl_pathv* points to a list containing the following:
 - 36280 a. Zero or more null pointers, as specified by GLOB_DOOFFS and *pglob->gl_offs*.
 - 36281 b. Pointers to the pathnames that were in the *pglob->gl_pathv* list before the call, in
 36282 the same order as before.
 - 36283 c. Pointers to the new pathnames generated by the second call, in the specified order.
 - 36284 4. The count returned in *pglob->gl_pathc* shall be the total number of pathnames from the
 36285 two calls.
 - 36286 5. The application can change any of the fields after a call to *glob()*. If it does, the
 36287 application shall reset them to the original value before a subsequent call, using the same
 36288 *pglob* value, to *globfree()* or *glob()* with the GLOB_APPEND flag.
- 36289 If, during the search, a directory is encountered that cannot be opened or read and *errfunc* is not
 36290 a null pointer, *glob()* calls (**errfunc()*) with two arguments:
- 36291 1. The *epath* argument is a pointer to the path that failed.
 - 36292 2. The *eerrno* argument is the value of *errno* from the failure, as set by *opendir()*, *readdir()*, or
 36293 *stat()*. (Other values may be used to report other errors not explicitly documented for
 36294 those functions.)
- 36295 If (**errfunc()*) is called and returns non-zero, or if the GLOB_ERR flag is set in *flags*, *glob()* shall
 36296 stop the scan and return GLOB_ABORTED after setting *gl_pathc* and *gl_pathv* in *pglob* to reflect
 36297 the paths already scanned. If GLOB_ERR is not set and either *errfunc* is a null pointer or
 36298 (**errfunc()*) returns 0, the error shall be ignored.
- 36299 The *glob()* function shall not fail because of large files.
- 36300 **RETURN VALUE**
- 36301 Upon successful completion, *glob()* shall return 0. The argument *pglob->gl_pathc* shall return the
 36302 number of matched pathnames and the argument *pglob->gl_pathv* shall contain a pointer to a
 36303 null-terminated list of matched and sorted pathnames. However, if *pglob->gl_pathc* is 0, the
 36304 content of *pglob->gl_pathv* is undefined.
- 36305 The *globfree()* function shall not return a value.
- 36306 If *glob()* terminates due to an error, it shall return one of the non-zero constants defined in
 36307 **<glob.h>**. The arguments *pglob->gl_pathc* and *pglob->gl_pathv* are still set as defined above.

ERRORS

The *glob()* function shall fail and return the corresponding value if:

36310 36311	GLOB_ABORTED	The scan was stopped because GLOB_ERR was set or (*errfunc()) returned non-zero.
36312 36313	GLOB_NOMATCH	The pattern does not match any existing pathname, and GLOB_NOCHECK was not set in flags.
36314	GLOB_NOSPACE	An attempt to allocate memory failed.

EXAMPLES

One use of the GLOB_DOOFFS flag is by applications that build an argument list for use with *execv()*, *execve()*, or *execvp()*. Suppose, for example, that an application wants to do the equivalent of:

```
ls -l *.c
```

but for some reason:

```
system("ls -l *.c")
```

is not acceptable. The application could obtain approximately the same result using the sequence:

```
globbuf.gl_offs = 2;
glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";
globbuf.gl_pathv[1] = "-l";
execvp("ls", &globbuf.gl_pathv[0]);
```

Using the same example:

```
ls -l *.c *.h
```

could be approximately simulated using GLOB_APPEND as follows:

```
globbuf.gl_offs = 2;
glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
glob("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
...
```

APPLICATION USAGE

This function is not provided for the purpose of enabling utilities to perform pathname expansion on their arguments, as this operation is performed by the shell, and utilities are explicitly not expected to redo this. Instead, it is provided for applications that need to do pathname expansion on strings obtained from other sources, such as a pattern typed by a user or read from a file.

If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch()*.

Note that *gl_pathc* and *gl_pathv* have meaning even if *glob()* fails. This allows *glob()* to report partial results in the event of an error. However, if *gl_pathc* is 0, *gl_pathv* is unspecified even if *glob()* did not return an error.

The GLOB_NOCHECK option could be used when an application wants to expand a pathname if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility might use this for option-arguments, for example.

The new pathnames generated by a subsequent call with GLOB_APPEND are not sorted together with the previous pathnames. This mirrors the way that the shell handles pathname

glob()

expansion when multiple expansions are done on a command line.

Applications that need tilde and parameter expansion should use *wordexp()*.

RATIONALE

It was claimed that the GLOB_DOOFFS flag is unnecessary because it could be simulated using:

```
new = (char **)malloc((n + pglob->gl_pathc + 1)
    * sizeof(char *));
(void) memcpy(new+n, pglob->gl_pathv,
    pglob->gl_pathc * sizeof(char *));
(void) memset(new, 0, n * sizeof(char *));
free(pglob->gl_pathv);
pglob->gl_pathv = new;
```

However, this assumes that the memory pointed to by *gl_pathv* is a block that was separately created using *malloc()*. This is not necessarily the case. An application should make no assumptions about how the memory referenced by fields in *pglob* was allocated. It might have been obtained from *malloc()* in a large chunk and then carved up within *glob()*, or it might have been created using a different memory allocator. It is not the intent of the standard developers to specify or imply how the memory used by *glob()* is managed.

The GLOB_APPEND flag would be used when an application wants to expand several different patterns into a single list.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fdopendir()*, *fnmatch()*, *fstatat()*, *readdir()*, Section 2.6

XBD Section 7.3.2 (on page 146), **<glob.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

Moved from POSIX2 C-language Binding to BASE.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

The **restrict** keyword is added to the *glob()* prototype for alignment with the ISO/IEC 9899:1999 standard.

NAME

gmtime, gmtime_r — convert a time value to a broken-down UTC time

SYNOPSIS

```
#include <time.h>

struct tm *gmtime(const time_t *timer);
struct tm *gmtime_r(const time_t *restrict timer,
struct tm *restrict result);
```

DESCRIPTION

For **gmtime()**: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The **gmtime()** function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time, expressed as Coordinated Universal Time (UTC).

The relationship between a time in seconds since the Epoch used as an argument to **gmtime()** and the **tm** structure (defined in the **<time.h>** header) is that the result shall be as specified in the expression given in the definition of seconds since the Epoch (see XBD Section 4.15, on page 113), where the names in the structure and in the expression correspond.

The same relationship shall apply for **gmtime_r()**.

The **gmtime()** function need not be thread-safe.

The **asctime()**, **ctime()**, **gmtime()**, and **localtime()** functions shall return values in one of two static objects: a broken-down time structure and an array of type **char**. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

The **gmtime_r()** function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time expressed as Coordinated Universal Time (UTC). The broken-down time is stored in the structure referred to by *result*. The **gmtime_r()** function shall also return the address of the same structure.

RETURN VALUE

Upon successful completion, the **gmtime()** function shall return a pointer to a **struct tm**. If an error is detected, **gmtime()** shall return a null pointer and set *errno* to indicate the error.

Upon successful completion, **gmtime_r()** shall return the address of the structure pointed to by the argument *result*. If an error is detected, **gmtime_r()** shall return a null pointer and set *errno* to indicate the error.

ERRORS

The **gmtime()** and **gmtime_r()** functions shall fail if:

[EOVERFLOW] The result cannot be represented.

gmtime()*System Interfaces*36419 **EXAMPLES**

36420 None.

36421 **APPLICATION USAGE**

36422 The *gmtime_r()* function is thread-safe and returns values in a user-supplied buffer instead of
 36423 possibly using a static data area that may be overwritten by each call.

36424 **RATIONALE**

36425 None.

36426 **FUTURE DIRECTIONS**

36427 None.

36428 **SEE ALSO**36429 *asctime(), clock(), ctime(), difftime(), localtime(), mktime(), strftime(), strptime(), time(), utime()*36430 XBD Section 4.15 (on page 113), *<time.h>*36431 **CHANGE HISTORY**

36432 First released in Issue 1. Derived from Issue 1 of the SVID.

36433 **Issue 5**

36434 A note indicating that the *gmtime()* function need not be reentrant is added to the
 36435 DESCRIPTION.

36436 The *gmtime_r()* function is included for alignment with the POSIX Threads Extension.36437 **Issue 6**36438 The *gmtime_r()* function is marked as part of the Thread-Safe Functions option.

36439 Extensions beyond the ISO C standard are marked.

36440 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
 36441 its avoidance of possibly using a static data area.

36442 The **restrict** keyword is added to the *gmtime_r()* prototype for alignment with the
 36443 ISO/IEC 9899:1999 standard.

36444 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/27 is applied, adding the [Eoverflow] error.
 36445

36446 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/48 is applied, updating the error handling
 36447 for *gmtime_r()*.

36448 **Issue 7**

36449 Austin Group Interpretation 1003.1-2001 #156 is applied.

36450 The *gmtime_r()* function is moved from the Thread-Safe Functions option to the Base.

NAME

grantpt — grant access to the slave pseudo-terminal device

SYNOPSIS

```
#include <stdlib.h>

int grantpt(int fildes);
```

DESCRIPTION

The *grantpt()* function shall change the mode and ownership of the slave pseudo-terminal device associated with its master pseudo-terminal counterpart. The *fildes* argument is a file descriptor that refers to a master pseudo-terminal device. The user ID of the slave shall be set to the real UID of the calling process and the group ID shall be set to an unspecified group ID. The permission mode of the slave pseudo-terminal shall be set to readable and writable by the owner, and writable by the group.

The behavior of the *grantpt()* function is unspecified if the application has installed a signal handler to catch SIGCHLD signals.

RETURN VALUE

Upon successful completion, *grantpt()* shall return 0; otherwise, it shall return -1 and set *errno* to indicate the error.

ERRORS

The *grantpt()* function may fail if:

[EBADF]	The <i>fildes</i> argument is not a valid open file descriptor.
[EINVAL]	The <i>fildes</i> argument is not associated with a master pseudo-terminal device.
[EACCES]	The corresponding slave pseudo-terminal device could not be accessed.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

open(), *ptsname()*, *unlockpt()*

XBD <stdlib.h>

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

hcreate()*System Interfaces*36489 **NAME**36490 **hcreate, hdestroy, hsearch** — manage hash search table36491 **SYNOPSIS**

```

36492 XSI      #include <search.h>
36493
36493      int hcreate(size_t nel);
36494      void hdestroy(void);
36495      ENTRY *hsearch(ENTRY item, ACTION action);

```

36496 **DESCRIPTION**36497 The *hcreate()*, *hdestroy()*, and *hsearch()* functions shall manage hash search tables.

36498 The *hcreate()* function shall allocate sufficient space for the table, and the application shall ensure it is called before *hsearch()* is used. The *nel* argument is an estimate of the maximum number of entries that the table shall contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

36502 The *hdestroy()* function shall dispose of the search table, and may be followed by another call to *hcreate()*. After the call to *hdestroy()*, the data can no longer be considered accessible.

36504 The *hsearch()* function is a hash-table search routine. It shall return a pointer into a hash table indicating the location at which an entry can be found. The *item* argument is a structure of type **ENTRY** (defined in the **<search.h>** header) containing two pointers: *item.key* points to the comparison key (a **char ***), and *item.data* (a **void ***) points to any other data to be associated with that key. The comparison function used by *hsearch()* is *strcmp()*. The *action* argument is a member of an enumeration type **ACTION** indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a null pointer.

36513 These functions need not be thread-safe.

36514 **RETURN VALUE**36515 The *hcreate()* function shall return 0 if it cannot allocate sufficient space for the table; otherwise, it shall return non-zero.36517 The *hdestroy()* function shall not return a value.36518 The *hsearch()* function shall return a null pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.36520 **ERRORS**36521 The *hcreate()* and *hsearch()* functions may fail if:36522 **[ENOMEM]** Insufficient storage space is available.36523 **EXAMPLES**

36524 The following example reads in strings followed by two numbers and stores them in a hash table, discarding duplicates. It then reads in strings and finds the matching entry in the hash table and prints it out.

```

36527 #include <stdio.h>
36528 #include <search.h>
36529 #include <string.h>
36530
36530 struct info {          /* This is the info stored in the table */
36531     int age, room;     /* other than the key. */
36532 };

```

```

36533     #define NUM_EMPL      5000      /* # of elements in search table. */
36534     int main(void)
36535     {
36536         char string_space[NUM_EMPL*20]; /* Space to store strings. */
36537         struct info info_space[NUM_EMPL]; /* Space to store employee info. */
36538         char *str_ptr = string_space; /* Next space in string_space. */
36539         struct info *info_ptr = info_space;
36540                                     /* Next space in info_space. */
36541         ENTRY item;
36542         ENTRY *found_item; /* Name to look for in table. */
36543         char name_to_find[30];
36544
36545         int i = 0;
36546
36547         /* Create table; no error checking is performed. */
36548         (void) hcreate(NUM_EMPL);
36549         while (scanf("%s%d%d", str_ptr, &info_ptr->age,
36550                     &info_ptr->room) != EOF && i++ < NUM_EMPL) {
36551
36552             /* Put information in structure, and structure in item. */
36553             item.key = str_ptr;
36554             item.data = info_ptr;
36555             str_ptr += strlen(str_ptr) + 1;
36556             info_ptr++;
36557
36558             /* Put item into table. */
36559             (void) hsearch(item, ENTER);
36560         }
36561
36562         /* Access table. */
36563         item.key = name_to_find;
36564         while (scanf("%s", item.key) != EOF) {
36565             if ((found_item = hsearch(item, FIND)) != NULL) {
36566
36567                 /* If item is in the table. */
36568                 (void)printf("found %s, age = %d, room = %d\n",
36569                             found_item->key,
36570                             ((struct info *)found_item->data)->age,
36571                             ((struct info *)found_item->data)->room);
36572             } else
36573                 (void)printf("no such employee %s\n", name_to_find);
36574         }
36575         return 0;
36576     }

```

APPLICATION USAGE

The *hcreate()* and *hsearch()* functions may use *malloc()* to allocate space.

RATIONALE

None.

FUTURE DIRECTIONS

None.

hcreate()*System Interfaces*36577 **SEE ALSO**36578 *bsearch()*, *lsearch()*, *malloc()*, *strcmp()*, *tdelete()*36579 XBD *<search.h>*36580 **CHANGE HISTORY**

36581 First released in Issue 1. Derived from Issue 1 of the SVID.

36582 **Issue 6**

36583 The normative text is updated to avoid use of the term “must” for application requirements.

36584 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

36585 **Issue 7**

36586 Austin Group Interpretation 1003.1-2001 #156 is applied.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

htonl, htons, ntohl, ntohs — convert values between host and network byte order

SYNOPSIS

```
#include <arpa/inet.h>

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

DESCRIPTION

These functions shall convert 16-bit and 32-bit quantities between network byte order and host byte order.

On some implementations, these functions are defined as macros.

The `uint32_t` and `uint16_t` types are defined in `<inttypes.h>`.

RETURN VALUE

The `htonl()` and `htons()` functions shall return the argument value converted from host to network byte order.

The `ntohl()` and `ntohs()` functions shall return the argument value converted from network to host byte order.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

These functions are most often used in conjunction with IPv4 addresses and ports as returned by `gethostent()` and `getservent()`.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`endhostent()`, `endservent()`

XBD `<arpa/inet.h>`, `<inttypes.h>`

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

hypot()*System Interfaces*36621 **NAME**36622 **hypot, hypotf, hypotl** — Euclidean distance function36623 **SYNOPSIS**36624 `#include <math.h>`36625 `double hypot(double x, double y);`36626 `float hypotf(float x, float y);`36627 `long double hypotl(long double x, long double y);`36628 **DESCRIPTION**

36629 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 36630 conflict between the requirements described here and the ISO C standard is unintentional. This
 36631 volume of POSIX.1-2008 defers to the ISO C standard.

36632 These functions shall compute the value of the square root of x^2+y^2 without undue overflow or
 36633 underflow.

36634 An application wishing to check for error situations should set *errno* to zero and call
 36635 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 36636 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 36637 zero, an error has occurred.

36638 **RETURN VALUE**

36639 Upon successful completion, these functions shall return the length of the hypotenuse of a right-
 36640 angled triangle with sides of length *x* and *y*.

36641 If the correct value would cause overflow, a range error shall occur and *hypot()*, *hypotf()*, and
 36642 *hypotl()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 36643 respectively.

36644 MX If *x* or *y* is $\pm\text{Inf}$, $\pm\text{Inf}$ shall be returned (even if one of *x* or *y* is NaN).

36645 If *x* or *y* is NaN, and the other is not $\pm\text{Inf}$, a NaN shall be returned.

36646 If both arguments are subnormal and the correct result is subnormal, a range error may occur
 36647 and the correct result is returned.

36648 **ERRORS**

36649 These functions shall fail if:

36650 Range Error The result overflows.

36651 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 36652 then *errno* shall be set to [ERANGE]. If the integer expression
 36653 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 36654 floating-point exception shall be raised.

36655 These functions may fail if:

36656 MX Range Error The result underflows.

36657 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 36658 then *errno* shall be set to [ERANGE]. If the integer expression
 36659 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 36660 floating-point exception shall be raised.

EXAMPLES

See the EXAMPLES section in *atan2()*.

APPLICATION USAGE

hypot(x,y), *hypot(y,x)*, and *hypot(x, -y)* are equivalent.

hypot(x, ±0) is equivalent to *fabs(x)*.

Underflow only happens when both *x* and *y* are subnormal and the (inexact) result is also subnormal.

These functions take precautions against overflow during intermediate steps of the computation.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atan2(), *feclearexcept()*, *fetestexcept()*, *isnan()*, *sqrt()*

XBD Section 4.19 (on page 116), <math.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *hypot()* function is no longer marked as an extension.

The *hypotf()* and *hypotl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/49 is applied, updating the EXAMPLES section.

iconv()*System Interfaces*36693 **NAME**36694 `iconv` — codeset conversion function36695 **SYNOPSIS**

```

36696 #include <iconv.h>
36697
36697 size_t iconv(iconv_t cd, char **restrict inbuf,
36698             size_t *restrict inbytesleft, char **restrict outbuf,
36699             size_t *restrict outbytesleft);

```

36700 **DESCRIPTION**

36701 The `iconv()` function shall convert the sequence of characters from one codeset, in the array
 36702 specified by `inbuf`, into a sequence of corresponding characters in another codeset, in the array
 36703 specified by `outbuf`. The codesets are those specified in the `iconv_open()` call that returned the
 36704 conversion descriptor, `cd`. The `inbuf` argument points to a variable that points to the first
 36705 character in the input buffer and `inbytesleft` indicates the number of bytes to the end of the buffer
 36706 to be converted. The `outbuf` argument points to a variable that points to the first available byte in
 36707 the output buffer and `outbytesleft` indicates the number of the available bytes to the end of the
 36708 buffer.

36709 For state-dependent encodings, the conversion descriptor `cd` is placed into its initial shift state by
 36710 a call for which `inbuf` is a null pointer, or for which `inbuf` points to a null pointer. When `iconv()`
 36711 is called in this way, and if `outbuf` is not a null pointer or a pointer to a null pointer, and `outbytesleft`
 36712 points to a positive value, `iconv()` shall place, into the output buffer, the byte sequence to change
 36713 the output buffer to its initial shift state. If the output buffer is not large enough to hold the
 36714 entire reset sequence, `iconv()` shall fail and set `errno` to [E2BIG]. Subsequent calls with `inbuf` as
 36715 other than a null pointer or a pointer to a null pointer cause the conversion to take place from
 36716 the current state of the conversion descriptor.

36717 If a sequence of input bytes does not form a valid character in the specified codeset, conversion
 36718 shall stop after the previous successfully converted character. If the input buffer ends with an
 36719 incomplete character or shift sequence, conversion shall stop after the previous successfully
 36720 converted bytes. If the output buffer is not large enough to hold the entire converted input,
 36721 conversion shall stop just prior to the input bytes that would cause the output buffer to
 36722 overflow. The variable pointed to by `inbuf` shall be updated to point to the byte following the last
 36723 byte successfully used in the conversion. The value pointed to by `inbytesleft` shall be
 36724 decremented to reflect the number of bytes still not converted in the input buffer. The variable
 36725 pointed to by `outbuf` shall be updated to point to the byte following the last byte of converted
 36726 output data. The value pointed to by `outbytesleft` shall be decremented to reflect the number of
 36727 bytes still available in the output buffer. For state-dependent encodings, the conversion
 36728 descriptor shall be updated to reflect the shift state in effect at the end of the last successfully
 36729 converted byte sequence.

36730 If `iconv()` encounters a character in the input buffer that is valid, but for which an identical
 36731 character does not exist in the target codeset, `iconv()` shall perform an implementation-defined
 36732 conversion on this character.

36733 **RETURN VALUE**

36734 The `iconv()` function shall update the variables pointed to by the arguments to reflect the extent
 36735 of the conversion and return the number of non-identical conversions performed. If the entire
 36736 string in the input buffer is converted, the value pointed to by `inbytesleft` shall be 0. If the input
 36737 conversion is stopped due to any conditions mentioned above, the value pointed to by `inbytesleft`
 36738 shall be non-zero and `errno` shall be set to indicate the condition. If an error occurs, `iconv()` shall
 36739 return `(size_t)-1` and set `errno` to indicate the error.

ERRORS

The *iconv()* function shall fail if:

- [EILSEQ] Input conversion stopped due to an input byte that does not belong to the input codeset.
- [E2BIG] Input conversion stopped due to lack of space in the output buffer.
- [EINVAL] Input conversion stopped due to an incomplete character or shift sequence at the end of the input buffer.

The *iconv()* function may fail if:

- [EBADF] The *cd* argument is not a valid open conversion descriptor.

EXAMPLES

None.

APPLICATION USAGE

The *inbuf* argument indirectly points to the memory area which contains the conversion input data. The *outbuf* argument indirectly points to the memory area which is to contain the result of the conversion. The objects indirectly pointed to by *inbuf* and *outbuf* are not restricted to containing data that is directly representable in the ISO C standard language **char** data type. The type of *inbuf* and *outbuf*, **char ****, does not imply that the objects pointed to are interpreted as null-terminated C strings or arrays of characters. Any interpretation of a byte sequence that represents a character in a given character set encoding scheme is done internally within the codeset converters. For example, the area pointed to indirectly by *inbuf* and/or *outbuf* can contain all zero octets that are not interpreted as string terminators but as coded character data according to the respective codeset encoding scheme. The type of the data (**char**, **short**, **long**, and so on) read or stored in the objects is not specified, but may be inferred for both the input and output data by the converters determined by the *fromcode* and *toencode* arguments of *iconv_open()*.

Regardless of the data type inferred by the converter, the size of the remaining space in both input and output objects (the *inbytesleft* and *outbytesleft* arguments) is always measured in bytes.

For implementations that support the conversion of state-dependent encodings, the conversion descriptor must be able to accurately reflect the shift-state in effect at the end of the last successful conversion. It is not required that the conversion descriptor itself be updated, which would require it to be a pointer type. Thus, implementations are free to implement the descriptor as a handle (other than a pointer type) by which the conversion information can be accessed and updated.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iconv_open(), *iconv_close()*, *mbsrtowcs()*

XBD **<iconv.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the HP-UX Manual.

iconv()*System Interfaces*36781 **Issue 6**36782 The SYNOPSIS has been corrected to align with the **<iconv.h>** reference page.36783 The **restrict** keyword is added to the *iconv()* prototype for alignment with the
36784 ISO/IEC 9899:1999 standard.36785 **Issue 7**36786 The *iconv()* function is moved from the XSI option to the Base.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

`iconv_close` — codeset conversion deallocation function

SYNOPSIS

```
#include <iconv.h>
```

```
int iconv_close(iconv_t cd);
```

DESCRIPTION

The `iconv_close()` function shall deallocate the conversion descriptor `cd` and all other associated resources allocated by `iconv_open()`.

If a file descriptor is used to implement the type **iconv_t**, that file descriptor shall be closed.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and `errno` set to indicate the error.

ERRORS

The `iconv_close()` function may fail if:

[EBADF] The conversion descriptor is invalid.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`iconv()`, `iconv_open()`

XBD **<iconv.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the HP-UX Manual.

Issue 7

The `iconv_close()` function is moved from the XSI option to the Base.

iconv_open()*System Interfaces*36817 **NAME**36818 `iconv_open` — codeset conversion allocation function36819 **SYNOPSIS**36820 `#include <iconv.h>`36821 `iconv_t iconv_open(const char *tocode, const char *fromcode);`36822 **DESCRIPTION**

36823 The `iconv_open()` function shall return a conversion descriptor that describes a conversion from
 36824 the codeset specified by the string pointed to by the *fromcode* argument to the codeset specified
 36825 by the string pointed to by the *tocode* argument. For state-dependent encodings, the conversion
 36826 descriptor shall be in a codeset-dependent initial shift state, ready for immediate use with
 36827 `iconv()`.

36828 Settings of *fromcode* and *tocode* and their permitted combinations are implementation-defined.

36829 A conversion descriptor shall remain valid until it is closed by `iconv_close()` or an implicit close.

36830 If a file descriptor is used to implement conversion descriptors, the `FD_CLOEXEC` flag shall be
 36831 set; see `<fcntl.h>`.

36832 **RETURN VALUE**

36833 Upon successful completion, `iconv_open()` shall return a conversion descriptor for use on
 36834 subsequent calls to `iconv()`. Otherwise, `iconv_open()` shall return `(iconv_t)-1` and set `errno` to
 36835 indicate the error.

36836 **ERRORS**

36837 The `iconv_open()` function may fail if:

36838 [EMFILE] All file descriptors available to the process are currently open.

36839 [ENFILE] Too many files are currently open in the system.

36840 [ENOMEM] Insufficient storage space is available.

36841 [EINVAL] The conversion specified by *fromcode* and *tocode* is not supported by the
 36842 implementation.

36843 **EXAMPLES**

36844 None.

36845 **APPLICATION USAGE**

36846 Some implementations of `iconv_open()` use `malloc()` to allocate space for internal buffer areas.
 36847 The `iconv_open()` function may fail if there is insufficient storage space to accommodate these
 36848 buffers.

36849 Conforming applications must assume that conversion descriptors are not valid after a call to
 36850 one of the *exec* functions.

36851 Application developers should consult the system documentation to determine the supported
 36852 codesets and their naming schemes.

36853 **RATIONALE**

36854 None.

36855 **FUTURE DIRECTIONS**

36856 None.

36857 **SEE ALSO**36858 *iconv()*, *iconv_close()*

36859 XBD <fcntl.h>, <iconv.h>

36860 **CHANGE HISTORY**

36861 First released in Issue 4. Derived from the HP-UX Manual.

36862 **Issue 7**

36863 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

36864 The *iconv_open()* function is moved from the XSI option to the Base.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

if_freenameindex()

System Interfaces

36865 **NAME**36866 `if_freenameindex` — free memory allocated by `if_nameindex`36867 **SYNOPSIS**36868 `#include <net/if.h>`36869 `void if_freenameindex(struct if_nameindex *ptr);`36870 **DESCRIPTION**

36871 The `if_freenameindex()` function shall free the memory allocated by `if_nameindex()`. The `ptr`
36872 argument shall be a pointer that was returned by `if_nameindex()`. After `if_freenameindex()` has
36873 been called, the application shall not use the array of which `ptr` is the address.

36874 **RETURN VALUE**

36875 None.

36876 **ERRORS**

36877 No errors are defined.

36878 **EXAMPLES**

36879 None.

36880 **APPLICATION USAGE**

36881 None.

36882 **RATIONALE**

36883 None.

36884 **FUTURE DIRECTIONS**

36885 None.

36886 **SEE ALSO**36887 `getsockopt()`, `if_indextoname()`, `if_nameindex()`, `if_nametoindex()`, `setsockopt()`36888 XBD `<net/if.h>`36889 **CHANGE HISTORY**

36890 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

System Interfaces

if_indextoname()**NAME**

`if_indextoname` — map a network interface index to its corresponding name

SYNOPSIS

```
#include <net/if.h>
```

```
char *if_indextoname(unsigned ifindex, char *ifname);
```

DESCRIPTION

The `if_indextoname()` function shall map an interface index to its corresponding name.

When this function is called, *ifname* shall point to a buffer of at least {IF_NAMESIZE} bytes. The function shall place in this buffer the name of the interface with index *ifindex*.

RETURN VALUE

If *ifindex* is an interface index, then the function shall return the value supplied in *ifname*, which points to a buffer now containing the interface name. Otherwise, the function shall return a null pointer and set *errno* to indicate the error.

ERRORS

The `if_indextoname()` function shall fail if:

[ENXIO] The interface does not exist.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[*getsockopt\(\)*](#), [*if_freenameindex\(\)*](#), [*if_nameindex\(\)*](#), [*if_nametoindex\(\)*](#), [*setsockopt\(\)*](#)

XBD [*<net/if.h>*](#)

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/28 is applied, changing {IFNAMSIZ} to {IF_NAMESIZ} in the DESCRIPTION.

if_nameindex()

System Interfaces

36922 **NAME**36923 `if_nameindex` — return all network interface names and indexes36924 **SYNOPSIS**36925 `#include <net/if.h>`36926 `struct if_nameindex *if_nameindex(void);`36927 **DESCRIPTION**

36928 The `if_nameindex()` function shall return an array of `if_nameindex` structures, one structure per
 36929 interface. The end of the array is indicated by a structure with an `if_index` field of zero and an
 36930 `if_name` field of NULL.

36931 Applications should call `if_freenameindex()` to release the memory that may be dynamically
 36932 allocated by this function, after they have finished using it.

36933 **RETURN VALUE**

36934 An array of structures identifying local interfaces. A null pointer is returned upon an error, with
 36935 `errno` set to indicate the error.

36936 **ERRORS**36937 The `if_nameindex()` function may fail if:

36938 [ENOBUFFS] Insufficient resources are available to complete the function.

36939 **EXAMPLES**

36940 None.

36941 **APPLICATION USAGE**

36942 None.

36943 **RATIONALE**

36944 None.

36945 **FUTURE DIRECTIONS**

36946 None.

36947 **SEE ALSO**36948 `getsockopt()`, `if_freenameindex()`, `if_indextoname()`, `if_nametoindex()`, `setsockopt()`36949 XBD `<net/if.h>`36950 **CHANGE HISTORY**

36951 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

36952 NAME

36953 `if_nametoindex` — map a network interface name to its corresponding index

36954 SYNOPSIS

36955 `#include <net/if.h>`

36956 `unsigned if_nametoindex(const char *ifname);`

36957 DESCRIPTION

36958 The `if_nametoindex()` function shall return the interface index corresponding to name *ifname*.

36959 RETURN VALUE

36960 The corresponding index if *ifname* is the name of an interface; otherwise, zero.

36961 ERRORS

36962 No errors are defined.

36963 EXAMPLES

36964 None.

36965 APPLICATION USAGE

36966 None.

36967 RATIONALE

36968 None.

36969 FUTURE DIRECTIONS

36970 None.

36971 SEE ALSO

36972 `getsockopt()`, `if_freenameindex()`, `if_indextoname()`, `if_nameindex()`, `setsockopt()`

36973 XBD `<net/if.h>`

36974 CHANGE HISTORY

36975 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

ilogb()*System Interfaces***NAME**

ilogb, ilogbf, ilogbl — return an unbiased exponent

SYNOPSIS

```
#include <math.h>

int ilogb(double x);
int ilogbf(float x);
int ilogbl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall return the exponent part of their argument x . Formally, the return value is the integral part of $\log_r |x|$ as a signed integral value, for non-zero x , where r is the radix of the machine's floating-point arithmetic, which is the value of FLT_RADIX defined in `<float.h>`.

An application wishing to check for error situations should set `errno` to zero and call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the exponent part of x as a signed integer value. They are equivalent to calling the corresponding `logb()` function and casting the returned value to type `int`.

If x is 0, the value FP_ILOGB0 shall be returned. On XSI-conformant systems, a domain error shall occur;

otherwise, a domain error may occur.

If x is $\pm\text{Inf}$, the value {INT_MAX} shall be returned. On XSI-conformant systems, a domain error shall occur;

otherwise, a domain error may occur.

If x is a NaN, the value FP_ILOGBNAN shall be returned. On XSI-conformant systems, a domain error shall occur;

otherwise, a domain error may occur.

If the correct value is greater than {INT_MAX}, a domain error shall occur and an unspecified value shall be returned. On XSI-conformant systems, a domain error shall occur and {INT_MAX} shall be returned.

If the correct value is less than {INT_MIN}, a domain error shall occur and an unspecified value shall be returned. On XSI-conformant systems, a domain error shall occur and {INT_MIN} shall be returned.

ERRORS

These functions shall fail if:

Domain Error The correct value is not representable as an integer.

The x argument is zero, NaN, or $\pm\text{Inf}$.

If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero, then `errno` shall be set to [EDOM]. If the integer expression (`math_errhandling & MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception shall be raised.

37021 These functions may fail if:

37022 Domain Error The *x* argument is zero, NaN, or $\pm\text{Inf}$.

37023 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
37024 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* &
37025 MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
37026 shall be raised.

37027 EXAMPLES

37028 None.

37029 APPLICATION USAGE

37030 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
37031 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

37032 RATIONALE

37033 The errors come from taking the expected floating-point value and converting it to **int**, which is
37034 an invalid operation in IEEE Std 754-1985 (since overflow, infinity, and NaN are not
37035 representable in a type **int**), so should be a domain error.

37036 There are no known implementations that overflow. For overflow to happen, {INT_MAX} must
37037 be less than $\text{LDBL\_MAX\_EXP} \cdot \log_2(\text{FLT\_RADIX})$ or {INT_MIN} must be greater than
37038 $\text{LDBL\_MIN\_EXP} \cdot \log_2(\text{FLT\_RADIX})$ if subnormals are not supported, or {INT_MIN} must be
37039 greater than $(\text{LDBL\_MIN\_EXP} - \text{LDBL\_MANT\_DIG}) \cdot \log_2(\text{FLT\_RADIX})$ if subnormals are
37040 supported.

37041 FUTURE DIRECTIONS

37042 None.

37043 SEE ALSO

37044 [*feclearexcept\(\)*](#), [*fetestexcept\(\)*](#), [*logb\(\)*](#), [*scalbn\(\)*](#)

37045 XBD Section 4.19 (on page 116), [*<float.h>*](#), [*<math.h>*](#)

37046 CHANGE HISTORY

37047 First released in Issue 4, Version 2.

37048 Issue 5

37049 Moved from X/OPEN UNIX extension to BASE.

37050 Issue 6

37051 The *ilogb()* function is no longer marked as an extension.

37052 The *ilogbf()* and *ilogbl()* functions are added for alignment with the ISO/IEC 9899:1999
37053 standard.

37054 The RETURN VALUE section is revised for alignment with the ISO/IEC 9899:1999 standard.

37055 Functionality relating to the XSI option is marked.

37056 Issue 7

37057 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #48 (SD5-XSH-ERN-71), #49, and #79
37058 (SD5-XSH-ERN-72) are applied.

imaxabs()

System Interfaces

37059 **NAME**37060 `imaxabs` — return absolute value37061 **SYNOPSIS**37062 `#include <inttypes.h>`37063 `intmax_t imaxabs(intmax_t j);`37064 **DESCRIPTION**

37065 CX The functionality described on this reference page is aligned with the ISO C standard. Any
37066 conflict between the requirements described here and the ISO C standard is unintentional. This
37067 volume of POSIX.1-2008 defers to the ISO C standard.

37068 The `imaxabs()` function shall compute the absolute value of an integer *j*. If the result cannot be
37069 represented, the behavior is undefined.

37070 **RETURN VALUE**37071 The `imaxabs()` function shall return the absolute value.37072 **ERRORS**

37073 No errors are defined.

37074 **EXAMPLES**

37075 None.

37076 **APPLICATION USAGE**

37077 The absolute value of the most negative number cannot be represented in two's complement.

37078 **RATIONALE**

37079 None.

37080 **FUTURE DIRECTIONS**

37081 None.

37082 **SEE ALSO**37083 [`imaxdiv\(\)`](#)37084 XBD [`<inttypes.h>`](#)37085 **CHANGE HISTORY**

37086 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

37087 **NAME**37088 **imaxdiv** — return quotient and remainder37089 **SYNOPSIS**37090 `#include <inttypes.h>`37091 `imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);`37092 **DESCRIPTION**

37093 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 37094 conflict between the requirements described here and the ISO C standard is unintentional. This
 37095 volume of POSIX.1-2008 defers to the ISO C standard.

37096 The *imaxdiv()* function shall compute *numer* / *denom* and *numer* % *denom* in a single operation.37097 **RETURN VALUE**

37098 The *imaxdiv()* function shall return a structure of type **imaxdiv_t**, comprising both the quotient
 37099 and the remainder. The structure shall contain (in either order) the members *quot* (the quotient)
 37100 and *rem* (the remainder), each of which has type **intmax_t**.

37101 If either part of the result cannot be represented, the behavior is undefined.

37102 **ERRORS**

37103 No errors are defined.

37104 **EXAMPLES**

37105 None.

37106 **APPLICATION USAGE**

37107 None.

37108 **RATIONALE**

37109 None.

37110 **FUTURE DIRECTIONS**

37111 None.

37112 **SEE ALSO**37113 [*imaxabs\(\)*](#)37114 XBD [*<inttypes.h>*](#)37115 **CHANGE HISTORY**

37116 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

inet_addr()*System Interfaces*37117 **NAME**37118 `inet_addr, inet_ntoa` — IPv4 address manipulation37119 **SYNOPSIS**

```
37120 #include <arpa/inet.h>
37121 in_addr_t inet_addr(const char *cp);
37122 char *inet_ntoa(struct in_addr in);
```

37123 **DESCRIPTION**

37124 The `inet_addr()` function shall convert the string pointed to by `cp`, in the standard IPv4 dotted decimal notation, to an integer value suitable for use as an Internet address.

37126 The `inet_ntoa()` function shall convert the Internet host address specified by `in` to a string in the Internet standard dot notation.

37128 The `inet_ntoa()` function need not be thread-safe.

37129 All Internet addresses shall be returned in network order (bytes ordered from left to right).

37130 Values specified using IPv4 dotted decimal notation take one of the following forms:

37131 `a.b.c.d` When four parts are specified, each shall be interpreted as a byte of data and
37132 assigned, from left to right, to the four bytes of an Internet address.

37133 `a.b.c` When a three-part address is specified, the last part shall be interpreted as a 16-bit
37134 quantity and placed in the rightmost two bytes of the network address. This makes
37135 the three-part address format convenient for specifying Class B network addresses
37136 as "`128.net.host`".

37137 `a.b` When a two-part address is supplied, the last part shall be interpreted as a 24-bit
37138 quantity and placed in the rightmost three bytes of the network address. This
37139 makes the two-part address format convenient for specifying Class A network
37140 addresses as "`net.host`".

37141 `a` When only one part is given, the value shall be stored directly in the network
37142 address without any byte rearrangement.

37143 All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or
37144 hexadecimal, as specified in the ISO C standard (that is, a leading `0x` or `0X` implies hexadecimal;
37145 otherwise, a leading `'0'` implies octal; otherwise, the number is interpreted as decimal).

37146 **RETURN VALUE**

37147 Upon successful completion, `inet_addr()` shall return the Internet address. Otherwise, it shall
37148 return `(in_addr_t)(-1)`.

37149 The `inet_ntoa()` function shall return a pointer to the network address in Internet standard dot
37150 notation.

37151 **ERRORS**

37152 No errors are defined.

37153 **EXAMPLES**

37154 None.

37155 **APPLICATION USAGE**

37156 The return value of *inet_ntoa()* may point to static data that may be overwritten by subsequent
37157 calls to *inet_ntoa()*.

37158 **RATIONALE**

37159 None.

37160 **FUTURE DIRECTIONS**

37161 None.

37162 **SEE ALSO**37163 *endhostent()*, *endnetent()*

37164 XBD <arpa/inet.h>

37165 **CHANGE HISTORY**

37166 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37167 **Issue 7**

37168 Austin Group Interpretation 1003.1-2001 #156 is applied.

inet_ntop()

System Interfaces

37169 **NAME**37170 `inet_ntop, inet_pton` — convert IPv4 and IPv6 addresses between binary and text form37171 **SYNOPSIS**

```
37172 #include <arpa/inet.h>

37173 const char *inet_ntop(int af, const void *restrict src,
37174     char *restrict dst, socklen_t size);
37175 int inet_pton(int af, const char *restrict src, void *restrict dst);
```

37176 **DESCRIPTION**

37177 The `inet_ntop()` function shall convert a numeric address into a text string suitable for
 37178 IP6 presentation. The `af` argument shall specify the family of the address. This can be `AF_INET` or
 37179 `AF_INET6`. The `src` argument points to a buffer holding an IPv4 address if the `af` argument is
 37180 IP6 `AF_INET`, or an IPv6 address if the `af` argument is `AF_INET6`; the address must be in network
 37181 byte order. The `dst` argument points to a buffer where the function stores the resulting text string;
 37182 it shall not be NULL. The `size` argument specifies the size of this buffer, which shall be large
 37183 IP6 enough to hold the text string (`INET_ADDRSTRLEN` characters for IPv4,
 37184 `INET6_ADDRSTRLEN` characters for IPv6).

37185 The `inet_pton()` function shall convert an address in its standard text presentation form into its
 37186 IP6 numeric binary form. The `af` argument shall specify the family of the address. The `AF_INET` and
 37187 `AF_INET6` address families shall be supported. The `src` argument points to the string being
 37188 passed in. The `dst` argument points to a buffer into which the function stores the numeric
 37189 IP6 address; this shall be large enough to hold the numeric address (32 bits for `AF_INET`, 128 bits
 37190 for `AF_INET6`).

37191 If the `af` argument of `inet_pton()` is `AF_INET`, the `src` string shall be in the standard IPv4 dotted-
 37192 decimal form:

37193 `ddd.ddd.ddd.ddd`

37194 where "ddd" is a one to three digit decimal number between 0 and 255 (see `inet_addr()`). The
 37195 `inet_pton()` function does not accept other formats (such as the octal numbers, hexadecimal
 37196 numbers, and fewer than four numbers that `inet_addr()` accepts).

37197 IP6 If the `af` argument of `inet_pton()` is `AF_INET6`, the `src` string shall be in one of the following
 37198 standard IPv6 text forms:

- 37199 1. The preferred form is "`x:x:x:x:x:x:x:x`", where the 'x's are the hexadecimal values
 37200 of the eight 16-bit pieces of the address. Leading zeros in individual fields can be
 37201 omitted, but there shall be at least one numeral in every field.
- 37202 2. A string of contiguous zero fields in the preferred form can be shown as "`::`". The "`::`"
 37203 can only appear once in an address. Unspecified addresses ("`0:0:0:0:0:0:0:0`") may
 37204 be represented simply as "`::`".
- 37205 3. A third form that is sometimes more convenient when dealing with a mixed environment
 37206 of IPv4 and IPv6 nodes is "`x:x:x:x:x:x:d.d.d.d`", where the 'x's are the
 37207 hexadecimal values of the six high-order 16-bit pieces of the address, and the 'd's are the
 37208 decimal values of the four low-order 8-bit pieces of the address (standard IPv4
 37209 representation).

37210 **Note:** A more extensive description of the standard representations of IPv6 addresses can be found in
 37211 RFC 2373.

RETURN VALUE

The *inet_ntop()* function shall return a pointer to the buffer containing the text string if the conversion succeeds, and NULL otherwise, and set *errno* to indicate the error.

The *inet_pton()* function shall return 1 if the conversion succeeds, with the address pointed to by *dst* in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string or a valid IPv6 address string, or -1 with *errno* set to [EAFNOSUPPORT] if the *af* argument is unknown.

ERRORS

The *inet_ntop()* and *inet_pton()* functions shall fail if:

[EAFNOSUPPORT]

The *af* argument is invalid.

[ENOSPC]

The size of the *inet_ntop()* result buffer is inadequate.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD [<arpa/inet.h>](#)

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

IPv6 extensions are marked.

The **restrict** keyword is added to the *inet_ntop()* and *inet_pton()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/29 is applied, adding “the address must be in network byte order” to the end of the fourth sentence of the first paragraph in the DESCRIPTION.

initstate()*System Interfaces*37242 **NAME**37243 `initstate`, `random`, `setstate`, `srandom` — pseudo-random number functions37244 **SYNOPSIS**

```

37245 XSI      #include <stdlib.h>
37246
37246 char *initstate(unsigned seed, char *state, size_t size);
37247 long random(void);
37248 char *setstate(char *state);
37249 void srandom(unsigned seed);

```

37250 **DESCRIPTION**

37251 The `random()` function shall use a non-linear additive feedback random-number generator
 37252 employing a default state array size of 31 **long** integers to return successive pseudo-random
 37253 numbers in the range from 0 to $2^{31}-1$. The period of this random-number generator is
 37254 approximately $16 \times (2^{31}-1)$. The size of the state array determines the period of the random-
 37255 number generator. Increasing the state array size shall increase the period.

37256 With 256 bytes of state information, the period of the random-number generator shall be greater
 37257 than 2^{69} .

37258 Like `rand()`, `random()` shall produce by default a sequence of numbers that can be duplicated by
 37259 calling `srandom()` with 1 as the seed.

37260 The `srandom()` function shall initialize the current state array using the value of `seed`.

37261 The `initstate()` and `setstate()` functions handle restarting and changing random-number
 37262 generators. The `initstate()` function allows a state array, pointed to by the `state` argument, to be
 37263 initialized for future use. The `size` argument, which specifies the size in bytes of the state array,
 37264 shall be used by `initstate()` to decide what type of random-number generator to use; the larger
 37265 the state array, the more random the numbers. Values for the amount of state information are 8,
 37266 32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one
 37267 of these values. If `initstate()` is called with $8 \leq \text{size} < 32$, then `random()` shall use a simple linear
 37268 congruential random number generator. The `seed` argument specifies a starting point for the
 37269 random-number sequence and provides for restarting at the same point. The `initstate()` function
 37270 shall return a pointer to the previous state information array.

37271 If `initstate()` has not been called, then `random()` shall behave as though `initstate()` had been called
 37272 with `seed=1` and `size=128`.

37273 Once a state has been initialized, `setstate()` allows switching between state arrays. The array
 37274 defined by the `state` argument shall be used for further random-number generation until
 37275 `initstate()` is called or `setstate()` is called again. The `setstate()` function shall return a pointer to the
 37276 previous state array.

37277 **RETURN VALUE**

37278 If `initstate()` is called with `size` less than 8, it shall return NULL.

37279 The `random()` function shall return the generated pseudo-random number.

37280 The `srandom()` function shall not return a value.

37281 Upon successful completion, `initstate()` and `setstate()` shall return a pointer to the previous state
 37282 array; otherwise, a null pointer shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

After initialization, a state array can be restarted at a different point in one of two ways:

1. The *initstate()* function can be used, with the desired seed, state array, and size of the array.
2. The *setstate()* function, with the desired state, can be used, followed by *srandom()* with the desired seed. The advantage of using both of these functions is that the size of the state array does not have to be saved once it is initialized.

Although some implementations of *random()* have written messages to standard error, such implementations do not conform to POSIX.1-2008.

Issue 5 restored the historical behavior of this function.

Threaded applications should use *erand48()*, *nrand48()*, or *jrand48()* instead of *random()* when an independent random number sequence in multiple threads is required.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

drand48(), *rand()*

XBD *<stdlib.h>*

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

In the DESCRIPTION, the phrase “values smaller than 8” is replaced with “values greater than or equal to 8, or less than 32”, “*size*<8” is replaced with “*8*≤*size* <32”, and a new first paragraph is added to the RETURN VALUE section. A note is added to the APPLICATION USAGE indicating that these changes restore the historical behavior of the function.

Issue 6

In the DESCRIPTION, duplicate text “For values greater than or equal to 8...” is removed.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/30 is applied, removing *rand_r()* from the list of suggested functions in the APPLICATION USAGE section.

Issue 7

The type of the first argument to *setstate()* is changed from **const char *** to **char ***.

insque()*System Interfaces*37320 **NAME**37321 **insque, remque** — insert or remove an element in a queue37322 **SYNOPSIS**

```

37323 XSI      #include <search.h>
37324          void insque(void *element, void *pred);
37325          void remque(void *element);

```

37326 **DESCRIPTION**

37327 The *insque()* and *remque()* functions shall manipulate queues built from doubly-linked lists. The
 37328 queue can be either circular or linear. An application using *insque()* or *remque()* shall ensure it
 37329 defines a structure in which the first two members of the structure are pointers to the same type
 37330 of structure, and any further members are application-specific. The first member of the structure
 37331 is a forward pointer to the next entry in the queue. The second member is a backward pointer to
 37332 the previous entry in the queue. If the queue is linear, the queue is terminated with null
 37333 pointers. The names of the structure and of the pointer members are not subject to any special
 37334 restriction.

37335 The *insque()* function shall insert the element pointed to by *element* into a queue immediately
 37336 after the element pointed to by *pred*.

37337 The *remque()* function shall remove the element pointed to by *element* from a queue.

37338 If the queue is to be used as a linear list, invoking *insque(&element, NULL)*, where *element* is the
 37339 initial element of the queue, shall initialize the forward and backward pointers of *element* to null
 37340 pointers.

37341 If the queue is to be used as a circular list, the application shall ensure it initializes the forward
 37342 pointer and the backward pointer of the initial element of the queue to the element's own
 37343 address.

37344 **RETURN VALUE**

37345 The *insque()* and *remque()* functions do not return a value.

37346 **ERRORS**

37347 No errors are defined.

37348 **EXAMPLES**37349 **Creating a Linear Linked List**

37350 The following example creates a linear linked list.

```

37351 #include <search.h>
37352 ...
37353 struct myque element1;
37354 struct myque element2;
37355
37356 char *data1 = "DATA1";
37357 char *data2 = "DATA2";
37358 ...
37359 element1.data = data1;
37360 element2.data = data2;
37361
37362 insque (&element1, NULL);
37363 insque (&element2, &element1);

```

Creating a Circular Linked List

The following example creates a circular linked list.

```
#include <search.h>
...
struct myque element1;
struct myque element2;

char *data1 = "DATA1";
char *data2 = "DATA2";
...
element1.data = data1;
element2.data = data2;

element1.fwd = &element1;
element1.bck = &element1;

insque (&element2, &element1);
```

Removing an Element

The following example removes the element pointed to by *element1*.

```
#include <search.h>
...
struct myque element1;
...
remque (&element1);
```

APPLICATION USAGE

The historical implementations of these functions described the arguments as being of type **struct qelem *** rather than as being of type **void *** as defined here. In those implementations, **struct qelem** was commonly defined in **<search.h>** as:

```
struct qelem {
    struct qelem *q_forw;
    struct qelem *q_back;
};
```

Applications using these functions, however, were never able to use this structure directly since it provided no room for the actual data contained in the elements. Most applications defined structures that contained the two pointers as the initial elements and also provided space for, or pointers to, the object's data. Applications that used these functions to update more than one type of table also had the problem of specifying two or more different structures with the same name, if they literally used **struct qelem** as specified.

As described here, the implementations were actually expecting a structure type where the first two members were forward and backward pointers to structures. With C compilers that didn't provide function prototypes, applications used structures as specified in the DESCRIPTION above and the compiler did what the application expected.

If this method had been carried forward with an ISO C standard compiler and the historical function prototype, most applications would have to be modified to cast pointers to the structures actually used to be pointers to **struct qelem** to avoid compilation warnings. By specifying **void *** as the argument type, applications do not need to change (unless they specifically referenced **struct qelem** and depended on it being defined in **<search.h>**).

insque()*System Interfaces*37406 **RATIONALE**

37407 None.

37408 **FUTURE DIRECTIONS**

37409 None.

37410 **SEE ALSO**37411 XBD [<search.h>](#)37412 **CHANGE HISTORY**

37413 First released in Issue 4, Version 2.

37414 **Issue 5**

37415 Moved from X/OPEN UNIX extension to BASE.

37416 **Issue 6**

37417 The normative text is updated to avoid use of the term “must” for application requirements.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

ioctl — control a STREAMS device (**STREAMS**)

SYNOPSIS

```
#include <stropts.h>

int ioctl(int fildes, int request, ... /* arg */);
```

DESCRIPTION

The *ioctl()* function shall perform a variety of control functions on STREAMS devices. For non-STREAMS devices, the functions performed by this call are unspecified. The *request* argument and an optional third argument (with varying type) shall be passed to and interpreted by the appropriate part of the STREAM associated with *fildes*.

The *fildes* argument is an open file descriptor that refers to a device.

The *request* argument selects the control function to be performed and shall depend on the STREAMS device being addressed.

The *arg* argument represents additional information that is needed by this specific STREAMS device to perform the requested function. The type of *arg* depends upon the particular control request, but it shall be either an integer or a pointer to a device-specific data structure.

The *ioctl()* commands applicable to STREAMS, their arguments, and error conditions that apply to each individual command are described below.

The following *ioctl()* commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH Pushes the module whose name is pointed to by *arg* onto the top of the current STREAM, just below the STREAM head. It then calls the *open()* function of the newly-pushed module.

The *ioctl()* function with the I_PUSH command shall fail if:

[EINVAL] Invalid module name.

[ENXIO] Open function of new module failed.

[ENXIO] Hangup received on *fildes*.

I_POP Removes the module just below the STREAM head of the STREAM pointed to by *fildes*. The *arg* argument should be 0 in an I_POP request.

The *ioctl()* function with the I_POP command shall fail if:

[EINVAL] No module present in the STREAM.

[ENXIO] Hangup received on *fildes*.

I_LOOK Retrieves the name of the module just below the STREAM head of the STREAM pointed to by *fildes*, and places it in a character string pointed to by *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long, where FMNAMESZ is defined in **<stropts.h>**.

The *ioctl()* function with the I_LOOK command shall fail if:

[EINVAL] No module present in the STREAM.

I_FLUSH Flushes read and/or write queues, depending on the value of *arg*. Valid *arg* values are:

ioctl()*System Interfaces*

37458		FLUSHR	Flush all read queues.
37459		FLUSHW	Flush all write queues.
37460		FLUSHRW	Flush all read and all write queues.
37461		The <i>ioctl()</i> function with the <i>I_FLUSH</i> command shall fail if:	
37462		[EINVAL]	Invalid <i>arg</i> value.
37463		[EAGAIN] or [ENOSR]	Unable to allocate buffers for flush message.
37464			
37465		[ENXIO]	Hangup received on <i>fildev</i> .
37466	I_FLUSHBAND	Flushes a particular band of messages. The <i>arg</i> argument points to a bandinfo structure. The <i>bi_flag</i> member may be one of FLUSHR, FLUSHW, or FLUSHRW as described above. The <i>bi_pri</i> member determines the priority band to be flushed.	
37467			
37468			
37469			
37470	I_SETSIG	Requests that the STREAMS implementation send the SIGPOLL signal to the calling process when a particular event has occurred on the STREAM associated with <i>fildev</i> . I_SETSIG supports an asynchronous processing capability in STREAMS. The value of <i>arg</i> is a bitmask that specifies the events for which the process should be signaled. It is the bitwise-inclusive OR of any combination of the following constants:	
37471			
37472			
37473			
37474			
37475			
37476		S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
37477			
37478			
37479		S_RDBAND	A message with a non-zero priority band has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
37480			
37481			
37482		S_INPUT	A message, other than a high-priority message, has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
37483			
37484			
37485		S_HIPRI	A high-priority message is present on a STREAM head read queue. A signal shall be generated even if the message is of zero length.
37486			
37487			
37488		S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.
37489			
37490			
37491			
37492		S_WRNORM	Equivalent to S_OUTPUT.
37493		S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) priority data downstream.
37494			
37495			
37496			
37497		S_MSG	A STREAMS signal message that contains the SIGPOLL signal has reached the front of the STREAM head read queue.
37498			
37499			

System Interfaces

ioctl()

37500	S_ERROR	Notification of an error condition has reached the STREAM head.
37501		
37502	S_HANGUP	Notification of a hangup has reached the STREAM head.
37503	S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.
37504		
37505		
37506		If <i>arg</i> is 0, the calling process shall be unregistered and shall not receive further SIGPOLL signals for the stream associated with <i>fildev</i> .
37507		
37508		Processes that wish to receive SIGPOLL signals shall ensure that they explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same STREAM, each process shall be signaled when the event occurs.
37509		
37510		
37511		
37512		The <i>ioctl()</i> function with the I_SETSIG command shall fail if:
37513	[EINVAL]	The value of <i>arg</i> is invalid.
37514	[EINVAL]	The value of <i>arg</i> is 0 and the calling process is not registered to receive the SIGPOLL signal.
37515		
37516	[EAGAIN]	There were insufficient resources to store the signal request.
37517	I_GETSIG	Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask in an int pointed to by <i>arg</i> , where the events are those specified in the description of I_SETSIG above.
37518		
37519		
37520		
37521		The <i>ioctl()</i> function with the I_GETSIG command shall fail if:
37522	[EINVAL]	Process is not registered to receive the SIGPOLL signal.
37523	I_FIND	Compares the names of all modules currently present in the STREAM to the name pointed to by <i>arg</i> , and returns 1 if the named module is present in the STREAM or returns 0 if the named module is not present.
37524		
37525		
37526		The <i>ioctl()</i> function with the I_FIND command shall fail if:
37527	[EINVAL]	<i>arg</i> does not contain a valid module name.
37528	I_PEEK	Retrieves the information in the first message on the STREAM head read queue without taking the message off the queue. It is analogous to <i>getmsg()</i> except that this command does not remove the message from the queue. The <i>arg</i> argument points to a strpeek structure.
37529		
37530		
37531		
37532		The application shall ensure that the <i>maxlen</i> member in the ctlbuf and databuf structures is set to the number of bytes of control information and/or data information, respectively, to retrieve. The <i>flags</i> member may be marked RS_HIPRI or 0, as described by <i>getmsg()</i> . If the process sets <i>flags</i> to RS_HIPRI, for example, I_PEEK shall only look for a high-priority message on the STREAM head read queue.
37533		
37534		
37535		
37536		
37537		
37538		I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the STREAM head read queue, or if the RS_HIPRI flag was set in <i>flags</i> and a high-priority message was not present on the STREAM head read queue. It does not wait for a message to arrive. On return, ctlbuf specifies information in the control buffer, databuf specifies information in the data
37539		
37540		
37541		
37542		

ioctl()*System Interfaces*

37543		buffer, and <i>flags</i> contains the value RS_HIPRI or 0.
37544	I_SRDOPT	Sets the read mode using the value of the argument <i>arg</i> . Read modes are described in <i>read()</i> . Valid <i>arg</i> flags are:
37545		
37546	RNORM	Byte-stream mode, the default.
37547	RMSGD	Message-discard mode.
37548	RMSGN	Message-nondiscard mode.
37549		The bitwise-inclusive OR of RMSGD and RMSGN shall return [EINVAL]. The bitwise-inclusive OR of RNORM and either RMSGD or RMSGN shall result in the other flag overriding RNORM which is the default.
37550		
37551		
37552		In addition, treatment of control messages by the STREAM head may be changed by setting any of the following flags in <i>arg</i> :
37553		
37554	RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.
37555		
37556	RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .
37557		
37558	RPROTDIS	Discard the control part of a message, delivering any data portion, when a process issues a <i>read()</i> .
37559		
37560		The <i>ioctl()</i> function with the I_SRDOPT command shall fail if:
37561		[EINVAL] The <i>arg</i> argument is not valid.
37562	I_GRDOPT	Returns the current read mode setting, as described above, in an int pointed to by the argument <i>arg</i> . Read modes are described in <i>read()</i> .
37563		
37564	I_NREAD	Counts the number of data bytes in the data part of the first message on the STREAM head read queue and places this value in the int pointed to by <i>arg</i> . The return value for the command shall be the number of messages on the STREAM head read queue. For example, if 0 is returned in <i>arg</i> , but the <i>ioctl()</i> return value is greater than 0, this indicates that a zero-length message is next on the queue.
37565		
37566		
37567		
37568		
37569		
37570	I_FDINSERT	Creates a message from specified buffer(s), adds information about another STREAM, and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below. The <i>arg</i> argument points to a strfdinsert structure.
37571		
37572		
37573		
37574		
37575		The application shall ensure that the <i>len</i> member in the ctlbuf strbuf structure is set to the size of a t_uscalar_t plus the number of bytes of control information to be sent with the message. The <i>fildev</i> member specifies the file descriptor of the other STREAM, and the <i>offset</i> member, which must be suitably aligned for use as a t_uscalar_t , specifies the offset from the start of the control buffer where I_FDINSERT shall store a t_uscalar_t whose interpretation is specific to the STREAM end. The application shall ensure that the <i>len</i> member in the databuf strbuf structure is set to the number of bytes of data information to be sent with the message, or to 0 if no data part is to be sent.
37576		
37577		
37578		
37579		
37580		
37581		
37582		
37583		
37584		
37585		The <i>flags</i> member specifies the type of message to be created. A normal message is created if <i>flags</i> is set to 0, and a high-priority message is created if
37586		

37587		<i>flags</i> is set to RS_HIPRI. For non-priority messages, I_FDINSERT shall block if
37588		the STREAM write queue is full due to internal flow control conditions. For
37589		priority messages, I_FDINSERT does not block on this condition. For non-
37590		priority messages, I_FDINSERT does not block when the write queue is full
37591		and O_NONBLOCK is set. Instead, it fails and sets <i>errno</i> to [EAGAIN].
37592		I_FDINSERT also blocks, unless prevented by lack of internal resources,
37593		waiting for the availability of message blocks in the STREAM, regardless of
37594		priority or whether O_NONBLOCK has been specified. No partial message is
37595		sent.
37596		The <i>ioctl()</i> function with the I_FDINSERT command shall fail if:
37597	[EAGAIN]	A non-priority message is specified, the O_NONBLOCK
37598		flag is set, and the STREAM write queue is full due to
37599		internal flow control conditions.
37600	[EAGAIN] or [ENOSR]	
37601		Buffers cannot be allocated for the message that is to be
37602		created.
37603	[EINVAL]	One of the following:
37604		— The <i>fildev</i> member of the strfdinsert structure is not a
37605		valid, open STREAM file descriptor.
37606		— The size of a t_uscalar_t plus <i>offset</i> is greater than the
37607		<i>len</i> member for the buffer specified through ctlbuf .
37608		— The <i>offset</i> member does not specify a properly-aligned
37609		location in the data buffer.
37610		— An undefined value is stored in <i>flags</i> .
37611	[ENXIO]	Hangup received on the STREAM identified by either the
37612		<i>fildev</i> argument or the <i>fildev</i> member of the strfdinsert
37613		structure.
37614	[ERANGE]	The <i>len</i> member for the buffer specified through databuf
37615		does not fall within the range specified by the maximum
37616		and minimum packet sizes of the topmost STREAM
37617		module; or the <i>len</i> member for the buffer specified through
37618		databuf is larger than the maximum configured size of the
37619		data part of a message; or the <i>len</i> member for the buffer
37620		specified through ctlbuf is larger than the maximum
37621		configured size of the control part of a message.
37622	I_STR	Constructs an internal STREAMS <i>ioctl()</i> message from the data pointed to by
37623		<i>arg</i> , and sends that message downstream.
37624		This mechanism is provided to send <i>ioctl()</i> requests to downstream modules
37625		and drivers. It allows information to be sent with <i>ioctl()</i> , and returns to the
37626		process any information sent upstream by the downstream recipient. I_STR
37627		shall block until the system responds with either a positive or negative
37628		acknowledgement message, or until the request times out after some period of
37629		time. If the request times out, it shall fail with <i>errno</i> set to [ETIME].
37630		At most, one I_STR can be active on a STREAM. Further I_STR calls shall
37631		block until the active I_STR completes at the STREAM head. The default

ioctl()*System Interfaces*

37632		timeout interval for these requests is 15 seconds. The O_NONBLOCK flag has
37633		no effect on this call.
37634		To send requests downstream, the application shall ensure that <i>arg</i> points to a
37635		struct ioctl structure.
37636		The <i>ic_cmd</i> member is the internal <i>ioctl()</i> command intended for a
37637		downstream module or driver and <i>ic_timeout</i> is the number of seconds
37638		(−1=infinite, 0=use implementation-defined timeout interval, >0=as specified)
37639		an I_STR request shall wait for acknowledgement before timing out. <i>ic_len</i> is
37640		the number of bytes in the data argument, and <i>ic_dp</i> is a pointer to the data
37641		argument. The <i>ic_len</i> member has two uses: on input, it contains the length of
37642		the data argument passed in, and on return from the command, it contains the
37643		number of bytes being returned to the process (the buffer pointed to by <i>ic_dp</i>
37644		should be large enough to contain the maximum amount of data that any
37645		module or the driver in the STREAM can return).
37646		The STREAM head shall convert the information pointed to by the struct ioctl
37647		structure to an internal <i>ioctl()</i> command message and send it downstream.
37648		The <i>ioctl()</i> function with the I_STR command shall fail if:
37649	[EAGAIN] or [ENOSR]	
37650		Unable to allocate buffers for the <i>ioctl()</i> message.
37651	[EINVAL]	The <i>ic_len</i> member is less than 0 or larger than the
37652		maximum configured size of the data part of a message, or
37653		<i>ic_timeout</i> is less than −1.
37654	[ENXIO]	Hangup received on <i>fildev</i> .
37655	[ETIME]	A downstream <i>ioctl()</i> timed out before acknowledgement
37656		was received.
37657		An I_STR can also fail while waiting for an acknowledgement if a message
37658		indicating an error or a hangup is received at the STREAM head. In addition,
37659		an error code can be returned in the positive or negative acknowledgement
37660		message, in the event the <i>ioctl()</i> command sent downstream fails. For these
37661		cases, I_STR shall fail with <i>errno</i> set to the value in the message.
37662	I_SWROPT	Sets the write mode using the value of the argument <i>arg</i> . Valid bit settings for
37663		<i>arg</i> are:
37664	SNDZERO	Send a zero-length message downstream when a <i>write()</i> of 0
37665		bytes occurs. To not send a zero-length message when a
37666		<i>write()</i> of 0 bytes occurs, the application shall ensure that
37667		this bit is not set in <i>arg</i> (for example, <i>arg</i> would be set to 0).
37668		The <i>ioctl()</i> function with the I_SWROPT command shall fail if:
37669	[EINVAL]	<i>arg</i> is not the above value.
37670	I_GWROPT	Returns the current write mode setting, as described above, in the int that is
37671		pointed to by the argument <i>arg</i> .
37672	I_SENDFD	Creates a new reference to the open file description associated with the file
37673		descriptor <i>arg</i> , and writes a message on the STREAMS-based pipe <i>fildev</i>
37674		containing this reference, together with the user ID and group ID of the calling
37675		process.

37676		The <i>ioctl()</i> function with the <code>L_SENDFD</code> command shall fail if:
37677	[EAGAIN]	The sending STREAM is unable to allocate a message block
37678		to contain the file pointer; or the read queue of the receiving
37679		STREAM head is full and cannot accept the message sent by
37680		<code>L_SENDFD</code> .
37681	[EBADF]	The <i>arg</i> argument is not a valid, open file descriptor.
37682	[EINVAL]	The <i>fildev</i> argument is not connected to a STREAM pipe.
37683	[ENXIO]	Hangup received on <i>fildev</i> .
37684		The <i>ioctl()</i> function with the <code>L_SENDFD</code> command may fail if:
37685	[EINVAL]	The <i>arg</i> argument is equal to the <i>fildev</i> argument.
37686	<code>L_RECVFD</code>	Retrieves the reference to an open file description from a message written to a
37687		STREAMS-based pipe using the <code>L_SENDFD</code> command, and allocates a new file
37688		descriptor in the calling process that refers to this open file description. The
37689		<i>arg</i> argument is a pointer to a strrecvfd data structure as defined in
37690		<stropts.h> .
37691		The <i>fd</i> member is a file descriptor. The <i>uid</i> and <i>gid</i> members are the effective
37692		user ID and effective group ID, respectively, of the sending process.
37693		If <code>O_NONBLOCK</code> is not set, <code>L_RECVFD</code> shall block until a message is present
37694		at the STREAM head. If <code>O_NONBLOCK</code> is set, <code>L_RECVFD</code> shall fail with <i>errno</i>
37695		set to [EAGAIN] if no message is present at the STREAM head.
37696		If the message at the STREAM head is a message sent by an <code>L_SENDFD</code> , a new
37697		file descriptor shall be allocated for the open file descriptor referenced in the
37698		message. The new file descriptor is placed in the <i>fd</i> member of the strrecvfd
37699		structure pointed to by <i>arg</i> .
37700		The <i>ioctl()</i> function with the <code>L_RECVFD</code> command shall fail if:
37701	[EAGAIN]	A message is not present at the STREAM head read queue
37702		and the <code>O_NONBLOCK</code> flag is set.
37703	[EBADMSG]	The message at the STREAM head read queue is not a
37704		message containing a passed file descriptor.
37705	[EMFILE]	All file descriptors available to the process are currently
37706		open.
37707	[ENXIO]	Hangup received on <i>fildev</i> .
37708	<code>L_LIST</code>	Allows the process to list all the module names on the STREAM, up to and
37709		including the topmost driver name. If <i>arg</i> is a null pointer, the return value
37710		shall be the number of modules, including the driver, that are on the STREAM
37711		pointed to by <i>fildev</i> . This lets the process allocate enough space for the module
37712		names. Otherwise, it should point to a str_list structure.
37713		The <i>sl_nmods</i> member indicates the number of entries the process has
37714		allocated in the array. Upon return, the <i>sl_modlist</i> member of the str_list
37715		structure shall contain the list of module names, and the number of entries
37716		that have been filled into the <i>sl_modlist</i> array is found in the <i>sl_nmods</i> member
37717		(the number includes the number of modules including the driver). The return
37718		value from <i>ioctl()</i> shall be 0. The entries are filled in starting at the top of the

ioctl()*System Interfaces*

37719		STREAM and continuing downstream until either the end of the STREAM is reached, or the number of requested modules (<i>sl_nmods</i>) is satisfied.
37720		
37721		The <i>ioctl()</i> function with the <i>I_LIST</i> command shall fail if:
37722		[EINVAL] The <i>sl_nmods</i> member is less than 1.
37723		[EAGAIN] or [ENOSR]
37724		Unable to allocate buffers.
37725	<i>I_ATMARK</i>	Allows the process to see if the message at the head of the STREAM head read queue is marked by some module downstream. The <i>arg</i> argument determines how the checking is done when there may be multiple marked messages on the STREAM head read queue. It may take on the following values:
37726		
37727		
37728		
37729		ANYMARK Check if the message is marked.
37730		LASTMARK Check if the message is the last one marked on the queue.
37731		The bitwise-inclusive OR of the flags ANYMARK and LASTMARK is permitted.
37732		
37733		The return value shall be 1 if the mark condition is satisfied; otherwise, the value shall be 0.
37734		
37735		The <i>ioctl()</i> function with the <i>I_ATMARK</i> command shall fail if:
37736		[EINVAL] Invalid <i>arg</i> value.
37737	<i>I_CKBAND</i>	Checks if the message of a given priority band exists on the STREAM head read queue. This shall return 1 if a message of the given priority exists, 0 if no such message exists, or -1 on error. <i>arg</i> should be of type <i>int</i> .
37738		
37739		
37740		The <i>ioctl()</i> function with the <i>I_CKBAND</i> command shall fail if:
37741		[EINVAL] Invalid <i>arg</i> value.
37742	<i>I_GETBAND</i>	Returns the priority band of the first message on the STREAM head read queue in the integer referenced by <i>arg</i> .
37743		
37744		The <i>ioctl()</i> function with the <i>I_GETBAND</i> command shall fail if:
37745		[ENODATA] No message on the STREAM head read queue.
37746	<i>I_CANPUT</i>	Checks if a certain band is writable. <i>arg</i> is set to the priority band in question. The return value shall be 0 if the band is flow-controlled, 1 if the band is writable, or -1 on error.
37747		
37748		
37749		The <i>ioctl()</i> function with the <i>I_CANPUT</i> command shall fail if:
37750		[EINVAL] Invalid <i>arg</i> value.
37751	<i>I_SETCLTIME</i>	This request allows the process to set the time the STREAM head shall delay when a STREAM is closing and there is data on the write queues. Before closing each module or driver, if there is data on its write queue, the STREAM head shall delay for the specified amount of time to allow the data to drain. If, after the delay, data is still present, it shall be flushed. The <i>arg</i> argument is a pointer to an integer specifying the number of milliseconds to delay, rounded up to the nearest valid value. If <i>I_SETCLTIME</i> is not performed on a STREAM, an implementation-defined default timeout interval is used.
37752		
37753		
37754		
37755		
37756		
37757		
37758		

37759 The *ioctl()* function with the *I_SETCLTIME* command shall fail if:

37760 [EINVAL] Invalid *arg* value.

37761 *I_GETCLTIME* Returns the close time delay in the integer pointed to by *arg*.

37762 Multiplexed STREAMS Configurations

37763 The following commands are used for connecting and disconnecting multiplexed STREAMS
37764 configurations. These commands use an implementation-defined default timeout interval.

37765 *I_LINK* Connects two STREAMs, where *fildes* is the file descriptor of the STREAM
37766 connected to the multiplexing driver, and *arg* is the file descriptor of the
37767 STREAM connected to another driver. The STREAM designated by *arg* is
37768 connected below the multiplexing driver. *I_LINK* requires the multiplexing
37769 driver to send an acknowledgement message to the STREAM head regarding
37770 the connection. This call shall return a multiplexer ID number (an identifier
37771 used to disconnect the multiplexer; see *I_UNLINK*) on success, and *-1* on
37772 failure.

37773 The *ioctl()* function with the *I_LINK* command shall fail if:

37774 [ENXIO] Hangup received on *fildes*.

37775 [ETIME] Timeout before acknowledgement message was received at
37776 STREAM head.

37777 [EAGAIN] or [ENOSR]

37778 Unable to allocate STREAMS storage to perform the
37779 *I_LINK*.

37780 [EBADF] The *arg* argument is not a valid, open file descriptor.

37781 [EINVAL] The *fildes* argument does not support multiplexing; or *arg* is
37782 not a STREAM or is already connected downstream from a
37783 multiplexer; or the specified *I_LINK* operation would
37784 connect the STREAM head in more than one place in the
37785 multiplexed STREAM.

37786 An *I_LINK* can also fail while waiting for the multiplexing driver to
37787 acknowledge the request, if a message indicating an error or a hangup is
37788 received at the STREAM head of *fildes*. In addition, an error code can be
37789 returned in the positive or negative acknowledgement message. For these
37790 cases, *I_LINK* fails with *errno* set to the value in the message.

37791 *I_UNLINK* Disconnects the two STREAMs specified by *fildes* and *arg*. *fildes* is the file
37792 descriptor of the STREAM connected to the multiplexing driver. The *arg*
37793 argument is the multiplexer ID number that was returned by the *I_LINK*
37794 *ioctl()* command when a STREAM was connected downstream from the
37795 multiplexing driver. If *arg* is *MUXID_ALL*, then all STREAMs that were
37796 connected to *fildes* shall be disconnected. As in *I_LINK*, this command requires
37797 acknowledgement.

37798 The *ioctl()* function with the *I_UNLINK* command shall fail if:

37799 [ENXIO] Hangup received on *fildes*.

ioctl()*System Interfaces*

37800		[ETIME]	Timeout before acknowledgement message was received at
37801			STREAM head.
37802		[EAGAIN] or [ENOSR]	
37803			Unable to allocate buffers for the acknowledgement
37804			message.
37805		[EINVAL]	Invalid multiplexer ID number.
37806			An I_UNLINK can also fail while waiting for the multiplexing driver to
37807			acknowledge the request if a message indicating an error or a hangup is
37808			received at the STREAM head of <i>fildev</i> . In addition, an error code can be
37809			returned in the positive or negative acknowledgement message. For these
37810			cases, I_UNLINK shall fail with <i>errno</i> set to the value in the message.
37811	I_PLINK		Creates a <i>persistent connection</i> between two STREAMs, where <i>fildev</i> is the file
37812			descriptor of the STREAM connected to the multiplexing driver, and <i>arg</i> is the
37813			file descriptor of the STREAM connected to another driver. This call shall
37814			create a persistent connection which can exist even if the file descriptor <i>fildev</i>
37815			associated with the upper STREAM to the multiplexing driver is closed. The
37816			STREAM designated by <i>arg</i> gets connected via a persistent connection below
37817			the multiplexing driver. I_PLINK requires the multiplexing driver to send an
37818			acknowledgement message to the STREAM head. This call shall return a
37819			multiplexer ID number (an identifier that may be used to disconnect the
37820			multiplexer; see I_PUNLINK) on success, and -1 on failure.
37821			The <i>ioctl()</i> function with the I_PLINK command shall fail if:
37822		[ENXIO]	Hangup received on <i>fildev</i> .
37823		[ETIME]	Timeout before acknowledgement message was received at
37824			STREAM head.
37825		[EAGAIN] or [ENOSR]	
37826			Unable to allocate STREAMS storage to perform the
37827			I_PLINK.
37828		[EBADF]	The <i>arg</i> argument is not a valid, open file descriptor.
37829		[EINVAL]	The <i>fildev</i> argument does not support multiplexing; or <i>arg</i> is
37830			not a STREAM or is already connected downstream from a
37831			multiplexer; or the specified I_PLINK operation would
37832			connect the STREAM head in more than one place in the
37833			multiplexed STREAM.
37834			An I_PLINK can also fail while waiting for the multiplexing driver to
37835			acknowledge the request, if a message indicating an error or a hangup is
37836			received at the STREAM head of <i>fildev</i> . In addition, an error code can be
37837			returned in the positive or negative acknowledgement message. For these
37838			cases, I_PLINK shall fail with <i>errno</i> set to the value in the message.
37839	I_PUNLINK		Disconnects the two STREAMs specified by <i>fildev</i> and <i>arg</i> from a persistent
37840			connection. The <i>fildev</i> argument is the file descriptor of the STREAM
37841			connected to the multiplexing driver. The <i>arg</i> argument is the multiplexer ID
37842			number that was returned by the I_PLINK <i>ioctl()</i> command when a STREAM
37843			was connected downstream from the multiplexing driver. If <i>arg</i> is
37844			MUXID_ALL, then all STREAMs which are persistent connections to <i>fildev</i>
37845			shall be disconnected. As in I_PLINK, this command requires the multiplexing

37846	driver to acknowledge the request.
37847	The <i>ioctl()</i> function with the <i>I_PUNLINK</i> command shall fail if:
37848	[ENXIO] Hangup received on <i>fildev</i> .
37849	[ETIME] Timeout before acknowledgement message was received at
37850	STREAM head.
37851	[EAGAIN] or [ENOSR]
37852	Unable to allocate buffers for the acknowledgement
37853	message.
37854	[EINVAL] Invalid multiplexer ID number.
37855	An <i>I_PUNLINK</i> can also fail while waiting for the multiplexing driver to
37856	acknowledge the request if a message indicating an error or a hangup is
37857	received at the STREAM head of <i>fildev</i> . In addition, an error code can be
37858	returned in the positive or negative acknowledgement message. For these
37859	cases, <i>I_PUNLINK</i> shall fail with <i>errno</i> set to the value in the message.
37860	RETURN VALUE
37861	Upon successful completion, <i>ioctl()</i> shall return a value other than <i>-1</i> that depends upon the
37862	STREAMS device control function. Otherwise, it shall return <i>-1</i> and set <i>errno</i> to indicate the
37863	error.
37864	ERRORS
37865	Under the following general conditions, <i>ioctl()</i> shall fail if:
37866	[EBADF] The <i>fildev</i> argument is not a valid open file descriptor.
37867	[EINTR] A signal was caught during the <i>ioctl()</i> operation.
37868	[EINVAL] The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or
37869	indirectly) downstream from a multiplexer.
37870	If an underlying device driver detects an error, then <i>ioctl()</i> shall fail if:
37871	[EINVAL] The <i>request</i> or <i>arg</i> argument is not valid for this device.
37872	[EIO] Some physical I/O error has occurred.
37873	[ENOTTY] The file associated with the <i>fildev</i> argument is not a STREAMS device that
37874	accepts control functions.
37875	[ENXIO] The <i>request</i> and <i>arg</i> arguments are valid for this device driver, but the service
37876	requested cannot be performed on this particular sub-device.
37877	[ENODEV] The <i>fildev</i> argument refers to a valid STREAMS device, but the corresponding
37878	device driver does not support the <i>ioctl()</i> function.
37879	If a STREAM is connected downstream from a multiplexer, any <i>ioctl()</i> command except
37880	<i>I_UNLINK</i> and <i>I_PUNLINK</i> shall set <i>errno</i> to [EINVAL].

ioctl()*System Interfaces*37881 **EXAMPLES**

37882 None.

37883 **APPLICATION USAGE**

37884 The implementation-defined timeout interval for STREAMS has historically been 15 seconds.

37885 **RATIONALE**

37886 None.

37887 **FUTURE DIRECTIONS**37888 The *ioctl()* function may be removed in a future version.37889 **SEE ALSO**37890 Section 2.6 (on page 494), *close()*, *fcntl()*, *getmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *read()*,
37891 *sigaction()*, *write()*

37892 XBD <stropts.h>

37893 **CHANGE HISTORY**

37894 First released in Issue 4, Version 2.

37895 **Issue 5**

37896 Moved from X/OPEN UNIX extension to BASE.

37897 **Issue 6**37898 The Open Group Corrigendum U028/4 is applied, correcting text in the L_FDINSERT [EINVAL]
37899 case to refer to *ctlbuf*.

37900 This function is marked as part of the XSI STREAMS Option Group.

37901 The normative text is updated to avoid use of the term “must” for application requirements.

37902 **Issue 7**37903 Austin Group Interpretation 1003.1-2001 #155 is applied, adding a “may fail” [EINVAL] error
37904 condition for the L_SENDFD command.

37905 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

37906 The *ioctl()* function is marked obsolescent.

NAME

isalnum, isalnum_l — test for an alphanumeric character

SYNOPSIS

```
#include <ctype.h>

int isalnum(int c);
int isalnum_l(int c, locale_t locale);
```

DESCRIPTION

For *isalnum()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *isalnum()* and *isalnum_l()* functions shall test whether *c* is a character of class **alpha** or **digit** in the current locale of the process, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).

The *c* argument is an **int**, the value of which the application shall ensure is representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isalnum()* and *isalnum_l()* functions shall return non-zero if *c* is an alphanumeric character; otherwise, they shall return 0.

ERRORS

The *isalnum_l()* function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalpha(), *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

XBD Chapter 7 (on page 135), *<ctype.h>*, *<stdio.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

isalnum()*System Interfaces*37947 **Issue 7**

37948

37949

The *isalnum_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

isalpha, isalpha_l — test for an alphabetic character

SYNOPSIS

```
#include <ctype.h>

int isalpha(int c);
int isalpha_l(int c, locale_t locale);
```

DESCRIPTION

For *isalpha()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *isalpha()* and *isalpha_l()* functions shall test whether *c* is a character of class **alpha** in the current locale of the process, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).

The *c* argument is an **int**, the value of which the application shall ensure is representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isalpha()* and *isalpha_l()* functions shall return non-zero if *c* is an alphabetic character; otherwise, they shall return 0.

ERRORS

The *isalpha_l()* function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum(), *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

XBD Chapter 7 (on page 135), *<ctype.h>*, *<locale.h>*, *<stdio.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

isalpha()*System Interfaces*37990 **Issue 7**

37991 The *isalpha_l()* function is added from The Open Group Technical Standard, 2006, Extended API
37992 Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

37993 NAME

37994 isascii — test for a 7-bit US-ASCII character

37995 SYNOPSIS

```
37996 OB XSI  #include <ctype.h>
37997          int isascii(int c);
```

37998 DESCRIPTION

37999 The *isascii()* function shall test whether *c* is a 7-bit US-ASCII character code.

38000 The *isascii()* function is defined on all integer values.

38001 RETURN VALUE

38002 The *isascii()* function shall return non-zero if *c* is a 7-bit US-ASCII character code between 0 and
38003 octal 0177 inclusive; otherwise, it shall return 0.

38004 ERRORS

38005 No errors are defined.

38006 EXAMPLES

38007 None.

38008 APPLICATION USAGE

38009 The *isascii()* function cannot be used portably in a localized application.

38010 RATIONALE

38011 None.

38012 FUTURE DIRECTIONS

38013 The *isascii()* function may be removed in a future version.

38014 SEE ALSO

38015 XBD [<ctype.h>](#)

38016 CHANGE HISTORY

38017 First released in Issue 1. Derived from Issue 1 of the SVID.

38018 Issue 7

38019 The *isascii()* function is marked obsolescent.

isastream()

System Interfaces

38020 **NAME**38021 isastream — test a file descriptor (**STREAMS**)38022 **SYNOPSIS**

```
38023 OB XSR #include <stropts.h>
38024 int isastream(int fildes);
```

38025 **DESCRIPTION**

38026 The *isastream()* function shall test whether *fildes*, an open file descriptor, is associated with a
 38027 STREAMS-based file.

38028 **RETURN VALUE**

38029 Upon successful completion, *isastream()* shall return 1 if *fildes* refers to a STREAMS-based file
 38030 and 0 if not. Otherwise, *isastream()* shall return -1 and set *errno* to indicate the error.

38031 **ERRORS**

38032 The *isastream()* function shall fail if:

38033 [EBADF] The *fildes* argument is not a valid open file descriptor.

38034 **EXAMPLES**

38035 None.

38036 **APPLICATION USAGE**

38037 None.

38038 **RATIONALE**

38039 None.

38040 **FUTURE DIRECTIONS**

38041 The *isastream()* function may be removed in a future version.

38042 **SEE ALSO**

38043 XBD [<stropts.h>](#)

38044 **CHANGE HISTORY**

38045 First released in Issue 4, Version 2.

38046 **Issue 5**

38047 Moved from X/OPEN UNIX extension to BASE.

38048 **Issue 7**

38049 The *isastream()* function is marked obsolescent.

NAME

isatty — test for a terminal device

SYNOPSIS

```
#include <unistd.h>
```

```
int isatty(int fildes);
```

DESCRIPTION

The *isatty()* function shall test whether *fildes*, an open file descriptor, is associated with a terminal device.

RETURN VALUE

The *isatty()* function shall return 1 if *fildes* is associated with a terminal; otherwise, it shall return 0 and may set *errno* to indicate the error.

ERRORS

The *isatty()* function may fail if:

[EBADF] The *fildes* argument is not a valid open file descriptor.

[ENOTTY] The file associated with the *fildes* argument is not a terminal.

EXAMPLES

None.

APPLICATION USAGE

The *isatty()* function does not necessarily indicate that a human being is available for interaction via *fildes*. It is quite possible that non-terminal devices are connected to the communications line.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD [<unistd.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional setting of *errno* to indicate an error is added.
- The [EBADF] and [ENOTTY] optional error conditions are added.

Issue 7

SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

isblank()

System Interfaces

38086 **NAME**

38087 isblank, isblank_l — test for a blank character

38088 **SYNOPSIS**

```
38089       #include <ctype.h>
38090       int isblank(int c);
38091 CX     int isblank_l(int c, locale_t locale);
```

38092 **DESCRIPTION**

38093 CX For *isblank()*: The functionality described on this reference page is aligned with the ISO C
 38094 standard. Any conflict between the requirements described here and the ISO C standard is
 38095 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38096 CX The *isblank()* and *isblank_l()* functions shall test whether *c* is a character of class **blank** in the
 38097 CX current locale of the process, or in the locale represented by *locale*, respectively; see XBD
 38098 Chapter 7 (on page 135).

38099 The *c* argument is a type **int**, the value of which the application shall ensure is a character
 38100 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 38101 any other value, the behavior is undefined.

38102 **RETURN VALUE**

38103 CX The *isblank()* and *isblank_l()* functions shall return non-zero if *c* is a <blank>; otherwise, they
 38104 shall return 0.

38105 **ERRORS**

38106 The *isblank_l()* function may fail if:

38107 CX [EINVAL] *locale* is not a valid locale object handle.

38108 **EXAMPLES**

38109 None.

38110 **APPLICATION USAGE**

38111 To ensure applications portability, especially across natural languages, only these functions and
 38112 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38113 classification.

38114 **RATIONALE**

38115 None.

38116 **FUTURE DIRECTIONS**

38117 None.

38118 **SEE ALSO**

38119 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,
 38120 *isxdigit()*, *setlocale()*, *uselocale()*

38121 XBD Chapter 7 (on page 135), <ctype.h>, <locale.h>

38122 **CHANGE HISTORY**

38123 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38124 **Issue 7**

38125 The *isblank_l()* function is added from The Open Group Technical Standard, 2006, Extended API
 38126 Set Part 4.

NAME

iscntrl, iscntrl_l — test for a control character

SYNOPSIS

```
#include <ctype.h>

int iscntrl(int c);
int iscntrl_l(int c, locale_t locale);
```

DESCRIPTION

For *iscntrl()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *iscntrl()* and *iscntrl_l()* functions shall test whether *c* is a character of class **cntrl** in the current locale of the process, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).

The *c* argument is a type **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iscntrl()* and *iscntrl_l()* functions shall return non-zero if *c* is a control character; otherwise, they shall return 0.

ERRORS

The *iscntrl_l()* function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum(), *isalpha()*, *isblank()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

XBD Chapter 7 (on page 135), *<ctype.h>*, *<locale.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

iscntrl()*System Interfaces*38167 **Issue 7**

38168 The *iscntrl_l()* function is added from The Open Group Technical Standard, 2006, Extended API
38169 Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

isdigit()

NAME

isdigit, isdigit_l — test for a decimal digit

SYNOPSIS

```
#include <ctype.h>

int isdigit(int c);
int isdigit_l(int c, locale_t locale);
```

DESCRIPTION

For *isdigit()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *isdigit()* and *isdigit_l()* functions shall test whether *c* is a character of class **digit** in the current locale of the process, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isdigit()* and *isdigit_l()* functions shall return non-zero if *c* is a decimal digit; otherwise, they shall return 0.

ERRORS

The *isdigit_l()* function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum(), *isalpha()*, *isblank()*, *iscntrl()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*

XBD Chapter 7 (on page 135), *<ctype.h>*, *<locale.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

isdigit()*System Interfaces*38210 **Issue 7**

38211 The *isdigit_l()* function is added from The Open Group Technical Standard, 2006, Extended API
38212 Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

38213 NAME

38214 isfinite — test for finite value

38215 SYNOPSIS

38216 #include <math.h>

38217 int isfinite(real-floating x);

38218 DESCRIPTION

38219 CX The functionality described on this reference page is aligned with the ISO C standard. Any
38220 conflict between the requirements described here and the ISO C standard is unintentional. This
38221 volume of POSIX.1-2008 defers to the ISO C standard.

38222 The *isfinite()* macro shall determine whether its argument has a finite value (zero, subnormal, or
38223 normal, and not infinite or NaN). First, an argument represented in a format wider than its
38224 semantic type is converted to its semantic type. Then determination is based on the type of the
38225 argument.

38226 RETURN VALUE

38227 The *isfinite()* macro shall return a non-zero value if and only if its argument has a finite value.

38228 ERRORS

38229 No errors are defined.

38230 EXAMPLES

38231 None.

38232 APPLICATION USAGE

38233 None.

38234 RATIONALE

38235 None.

38236 FUTURE DIRECTIONS

38237 None.

38238 SEE ALSO

38239 *fpclassify()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*

38240 XBD <math.h>

38241 CHANGE HISTORY

38242 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

isgraph()

System Interfaces

38243 **NAME**38244 `isgraph, isgraph_l` — test for a visible character38245 **SYNOPSIS**

```
38246 #include <ctype.h>
38247 int isgraph(int c);
38248 CX int isgraph_l(int c, locale_t locale);
```

38249 **DESCRIPTION**

38250 CX For `isgraph()`: The functionality described on this reference page is aligned with the ISO C
 38251 standard. Any conflict between the requirements described here and the ISO C standard is
 38252 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38253 CX The `isgraph()` and `isgraph_l()` functions shall test whether `c` is a character of class **graph** in the
 38254 CX current locale of the process, or in the locale represented by `locale`, respectively; see XBD
 38255 Chapter 7 (on page 135).

38256 The `c` argument is an **int**, the value of which the application shall ensure is a character
 38257 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 38258 any other value, the behavior is undefined.

38259 **RETURN VALUE**

38260 CX The `isgraph()` and `isgraph_l()` functions shall return non-zero if `c` is a character with a visible
 38261 representation; otherwise, they shall return 0.

38262 **ERRORS**

38263 The `isgraph_l()` function may fail if:

38264 CX [EINVAL] `locale` is not a valid locale object handle.

38265 **EXAMPLES**

38266 None.

38267 **APPLICATION USAGE**

38268 To ensure applications portability, especially across natural languages, only these functions and
 38269 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38270 classification.

38271 **RATIONALE**

38272 None.

38273 **FUTURE DIRECTIONS**

38274 None.

38275 **SEE ALSO**

38276 `isalnum()`, `isalpha()`, `isblank()`, `iscntrl()`, `isdigit()`, `islower()`, `isprint()`, `ispunct()`, `isspace()`, `isupper()`,
 38277 `isxdigit()`, `setlocale()`, `uselocale()`

38278 XBD Chapter 7 (on page 135), `<ctype.h>`, `<locale.h>`

38279 **CHANGE HISTORY**

38280 First released in Issue 1. Derived from Issue 1 of the SVID.

38281 **Issue 6**

38282 The normative text is updated to avoid use of the term “must” for application requirements.

38283 **Issue 7**

38284 The *isgraph_l()* function is added from The Open Group Technical Standard, 2006, Extended API
38285 Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

isgreater()

System Interfaces

38286 **NAME**38287 `isgreater` — test if *x* greater than *y*38288 **SYNOPSIS**38289 `#include <math.h>`38290 `int isgreater(real-floating x, real-floating y);`38291 **DESCRIPTION**

38292 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 38293 conflict between the requirements described here and the ISO C standard is unintentional. This
 38294 volume of POSIX.1-2008 defers to the ISO C standard.

38295 The `isgreater()` macro shall determine whether its first argument is greater than its second
 38296 argument. The value of `isgreater(x, y)` shall be equal to $(x) > (y)$; however, unlike $(x) > (y)$,
 38297 `isgreater(x, y)` shall not raise the invalid floating-point exception when *x* and *y* are unordered.

38298 **RETURN VALUE**38299 Upon successful completion, the `isgreater()` macro shall return the value of $(x) > (y)$.38300 If *x* or *y* is NaN, 0 shall be returned.38301 **ERRORS**

38302 No errors are defined.

38303 **EXAMPLES**

38304 None.

38305 **APPLICATION USAGE**

38306 The relational and equality operators support the usual mathematical relationships between
 38307 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
 38308 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
 38309 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
 38310 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
 38311 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
 38312 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
 38313 indicates that the argument shall be an expression of **real-floating** type.

38314 **RATIONALE**

38315 None.

38316 **FUTURE DIRECTIONS**

38317 None.

38318 **SEE ALSO**38319 `isgreaterequal()`, `isless()`, `islessequal()`, `islessgreater()`, `isunordered()`38320 **XBD** `<math.h>`38321 **CHANGE HISTORY**

38322 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

NAME

isgreaterequal — test if *x* is greater than or equal to *y*

SYNOPSIS

```
#include <math.h>
```

```
int isgreaterequal(real-floating x, real-floating y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *isgreaterequal()* macro shall determine whether its first argument is greater than or equal to its second argument. The value of *isgreaterequal(x, y)* shall be equal to $(x) \geq (y)$; however, unlike $(x) \geq (y)$, *isgreaterequal(x, y)* shall not raise the invalid floating-point exception when *x* and *y* are unordered.

RETURN VALUE

Upon successful completion, the *isgreaterequal()* macro shall return the value of $(x) \geq (y)$.

If *x* or *y* is NaN, 0 shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isgreater(), *isless()*, *islessequal()*, *islessgreater()*, *isunordered()*

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

isinf()*System Interfaces*38361 **NAME**

38362 isinf — test for infinity

38363 **SYNOPSIS**

38364 #include <math.h>

38365 int isinf(real-floating x);

38366 **DESCRIPTION**

38367 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 38368 conflict between the requirements described here and the ISO C standard is unintentional. This
 38369 volume of POSIX.1-2008 defers to the ISO C standard.

38370 The *isinf()* macro shall determine whether its argument value is an infinity (positive or
 38371 negative). First, an argument represented in a format wider than its semantic type is converted
 38372 to its semantic type. Then determination is based on the type of the argument.

38373 **RETURN VALUE**38374 The *isinf()* macro shall return a non-zero value if and only if its argument has an infinite value.38375 **ERRORS**

38376 No errors are defined.

38377 **EXAMPLES**

38378 None.

38379 **APPLICATION USAGE**

38380 None.

38381 **RATIONALE**

38382 None.

38383 **FUTURE DIRECTIONS**

38384 None.

38385 **SEE ALSO**38386 *fpclassify()*, *isfinite()*, *isnan()*, *ishnormal()*, *signbit()*

38387 XBD <math.h>

38388 **CHANGE HISTORY**

38389 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

NAME

isless — test if *x* is less than *y*

SYNOPSIS

```
#include <math.h>
```

```
int isless(real-floating x, real-floating y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *isless()* macro shall determine whether its first argument is less than its second argument. The value of *isless(x, y)* shall be equal to $(x) < (y)$; however, unlike $(x) < (y)$, *isless(x, y)* shall not raise the invalid floating-point exception when *x* and *y* are unordered.

RETURN VALUE

Upon successful completion, the *isless()* macro shall return the value of $(x) < (y)$.

If *x* or *y* is NaN, 0 shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isgreater(), *isgreaterequal()*, *islessequal()*, *islessgreater()*, *isunordered()*

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

islessequal()

System Interfaces

38427 **NAME**38428 `islessequal` — test if *x* is less than or equal to *y*38429 **SYNOPSIS**38430 `#include <math.h>`38431 `int islessequal(real-floating x, real-floating y);`38432 **DESCRIPTION**

38433 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 38434 conflict between the requirements described here and the ISO C standard is unintentional. This
 38435 volume of POSIX.1-2008 defers to the ISO C standard.

38436 The `islessequal()` macro shall determine whether its first argument is less than or equal to its
 38437 second argument. The value of `islessequal(x, y)` shall be equal to $(x) \leq (y)$; however, unlike
 38438 $(x) \leq (y)$, `islessequal(x, y)` shall not raise the invalid floating-point exception when *x* and *y* are
 38439 unordered.

38440 **RETURN VALUE**38441 Upon successful completion, the `islessequal()` macro shall return the value of $(x) \leq (y)$.38442 If *x* or *y* is NaN, 0 shall be returned.38443 **ERRORS**

38444 No errors are defined.

38445 **EXAMPLES**

38446 None.

38447 **APPLICATION USAGE**

38448 The relational and equality operators support the usual mathematical relationships between
 38449 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
 38450 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
 38451 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
 38452 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
 38453 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
 38454 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
 38455 indicates that the argument shall be an expression of **real-floating** type.

38456 **RATIONALE**

38457 None.

38458 **FUTURE DIRECTIONS**

38459 None.

38460 **SEE ALSO**38461 `isgreater()`, `isgreaterequal()`, `isless()`, `islessgreater()`, `isunordered()`38462 XBD `<math.h>`38463 **CHANGE HISTORY**

38464 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

System Interfaces

islessgreater()**NAME**

islessgreater — test if *x* is less than or greater than *y*

SYNOPSIS

```
#include <math.h>
```

```
int islessgreater(real-floating x, real-floating y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *islessgreater()* macro shall determine whether its first argument is less than or greater than its second argument. The *islessgreater(x, y)* macro is similar to $(x) < (y) \parallel (x) > (y)$; however, *islessgreater(x, y)* shall not raise the invalid floating-point exception when *x* and *y* are unordered (nor shall it evaluate *x* and *y* twice).

RETURN VALUE

Upon successful completion, the *islessgreater()* macro shall return the value of $(x) < (y) \parallel (x) > (y)$.

If *x* or *y* is NaN, 0 shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isgreater(), *isgreaterequal()*, *isless()*, *islessequal()*, *isunordered()*

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

islower()

System Interfaces

38504 **NAME**

38505 islower, islower_l — test for a lowercase letter

38506 **SYNOPSIS**

```
38507     #include <ctype.h>
38508     int islower(int c);
38509 CX    int islower_l(int c, locale_t locale);
```

38510 **DESCRIPTION**

38511 CX For *islower()*: The functionality described on this reference page is aligned with the ISO C
 38512 standard. Any conflict between the requirements described here and the ISO C standard is
 38513 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38514 CX The *islower()* and *islower_l()* functions shall test whether *c* is a character of class **lower** in the
 38515 CX current locale of the process, or in the locale represented by *locale*, respectively; see XBD
 38516 Chapter 7 (on page 135).

38517 The *c* argument is an **int**, the value of which the application shall ensure is a character
 38518 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 38519 any other value, the behavior is undefined.

38520 **RETURN VALUE**

38521 CX The *islower()* and *islower_l()* functions shall return non-zero if *c* is a lowercase letter; otherwise,
 38522 they shall return 0.

38523 **ERRORS**

38524 The *islower_l()* function may fail if:

38525 CX [EINVAL] *locale* is not a valid locale object handle.

38526 **EXAMPLES**38527 **Testing for a Lowercase Letter**

38528 Two examples follow, the first using *islower()*, the second using multiple concurrent locales and
 38529 *islower_l()*.

38530 The examples test whether the value is a lowercase letter, based on the locale of the user, then
 38531 use it as part of a key value.

```
38532 /* Example 1 -- using islower() */
38533 #include <ctype.h>
38534 #include <stdlib.h>
38535 #include <locale.h>
38536 ...
38537 char *keystr;
38538 int elementlen, len;
38539 char c;
38540 ...
38541 setlocale(LC_ALL, "");
38542 ...
38543 len = 0;
38544 while (len < elementlen) {
38545     c = (char) (rand() % 256);
38546     ...
38547     if (islower(c))
```



```

38548         keystr[len++] = c;
38549     }
38550     ...
38551     /* Example 2 -- using islower_l() */
38552     #include <ctype.h>
38553     #include <stdlib.h>
38554     #include <locale.h>
38555     ...
38556     char *keystr;
38557     int elementlen, len;
38558     char c;
38559     ...
38560     locale_t loc = newlocale (LC_ALL_MASK, "", (locale_t) 0);
38561     ...
38562     len = 0;
38563     while (len < elementlen) {
38564         c = (char) (rand() % 256);
38565         ...
38566         if (islower_l(c, loc))
38567             keystr[len++] = c;
38568     }
38569     ...

```

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum(), *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

XBD Chapter 7 (on page 135), *<ctype.h>*, *<locale.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

An example is added.

Issue 7

The *islower_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

isnan()*System Interfaces*38590 **NAME**38591 `isnan` — test for a NaN38592 **SYNOPSIS**38593 `#include <math.h>`38594 `int isnan(real-floating x);`38595 **DESCRIPTION**

38596 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 38597 conflict between the requirements described here and the ISO C standard is unintentional. This
 38598 volume of POSIX.1-2008 defers to the ISO C standard.

38599 The `isnan()` macro shall determine whether its argument value is a NaN. First, an argument
 38600 represented in a format wider than its semantic type is converted to its semantic type. Then
 38601 determination is based on the type of the argument.

38602 **RETURN VALUE**38603 The `isnan()` macro shall return a non-zero value if and only if its argument has a NaN value.38604 **ERRORS**

38605 No errors are defined.

38606 **EXAMPLES**

38607 None.

38608 **APPLICATION USAGE**

38609 None.

38610 **RATIONALE**

38611 None.

38612 **FUTURE DIRECTIONS**

38613 None.

38614 **SEE ALSO**38615 `fpclassify()`, `isfinite()`, `isinf()`, `isnormal()`, `signbit()`38616 XBD `<math.h>`38617 **CHANGE HISTORY**

38618 First released in Issue 3.

38619 **Issue 5**

38620 The DESCRIPTION is updated to indicate the return value when NaN is not supported. This
 38621 text was previously published in the APPLICATION USAGE section.

38622 **Issue 6**

38623 Re-written for alignment with the ISO/IEC 9899:1999 standard.

38624 NAME

38625 isnormal — test for a normal value

38626 SYNOPSIS

38627 #include <math.h>

38628 int isnormal(real-floating x);

38629 DESCRIPTION

38630 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 38631 conflict between the requirements described here and the ISO C standard is unintentional. This
 38632 volume of POSIX.1-2008 defers to the ISO C standard.

38633 The *isnormal()* macro shall determine whether its argument value is normal (neither zero,
 38634 subnormal, infinite, nor NaN). First, an argument represented in a format wider than its
 38635 semantic type is converted to its semantic type. Then determination is based on the type of the
 38636 argument.

38637 RETURN VALUE

38638 The *isnormal()* macro shall return a non-zero value if and only if its argument has a normal
 38639 value.

38640 ERRORS

38641 No errors are defined.

38642 EXAMPLES

38643 None.

38644 APPLICATION USAGE

38645 None.

38646 RATIONALE

38647 None.

38648 FUTURE DIRECTIONS

38649 None.

38650 SEE ALSO

38651 [*fpclassify\(\)*](#), [*isfinite\(\)*](#), [*isinf\(\)*](#), [*isnan\(\)*](#), [*signbit\(\)*](#)

38652 XBD [**<math.h>**](#)

38653 CHANGE HISTORY

38654 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

isprint()

System Interfaces

38655 **NAME**38656 `isprint, isprint_l` — test for a printable character38657 **SYNOPSIS**

```
38658     #include <ctype.h>
38659     int isprint(int c);
38660 CX    int isprint_l(int c, locale_t locale);
```

38661 **DESCRIPTION**

38662 CX For `isprint()`: The functionality described on this reference page is aligned with the ISO C
 38663 standard. Any conflict between the requirements described here and the ISO C standard is
 38664 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38665 CX The `isprint()` and `isprint_l()` functions shall test whether `c` is a character of class **print** in the
 38666 CX current locale of the process, or in the locale represented by `locale`, respectively; see XBD
 38667 Chapter 7 (on page 135).

38668 The `c` argument is an **int**, the value of which the application shall ensure is a character
 38669 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 38670 any other value, the behavior is undefined.

38671 **RETURN VALUE**

38672 CX The `isprint()` and `isprint_l()` functions shall return non-zero if `c` is a printable character;
 38673 otherwise, they shall return 0.

38674 **ERRORS**

38675 The `isprint_l()` function may fail if:

38676 CX [EINVAL] `locale` is not a valid locale object handle.

38677 **EXAMPLES**

38678 None.

38679 **APPLICATION USAGE**

38680 To ensure applications portability, especially across natural languages, only these functions and
 38681 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38682 classification.

38683 **RATIONALE**

38684 None.

38685 **FUTURE DIRECTIONS**

38686 None.

38687 **SEE ALSO**

38688 `isalnum()`, `isalpha()`, `isblank()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `ispunct()`, `isspace()`, `isupper()`,
 38689 `isxdigit()`, `setlocale()`, `uselocale()`

38690 XBD Chapter 7 (on page 135), `<ctype.h>`, `<locale.h>`

38691 **CHANGE HISTORY**

38692 First released in Issue 1. Derived from Issue 1 of the SVID.

38693 **Issue 6**

38694 The normative text is updated to avoid use of the term “must” for application requirements.

38695 **Issue 7**

38696 The *isprint_l()* function is added from The Open Group Technical Standard, 2006, Extended API
38697 Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

ispunct()*System Interfaces*38698 **NAME**

38699 ispunct, ispunct_l — test for a punctuation character

38700 **SYNOPSIS**

38701 #include <ctype.h>

38702 int ispunct(int c);

38703 CX int ispunct_l(int c, locale_t locale);

38704 **DESCRIPTION**38705 CX For *ispunct()*: The functionality described on this reference page is aligned with the ISO C
38706 standard. Any conflict between the requirements described here and the ISO C standard is
38707 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.38708 CX The *ispunct()* and *ispunct_l()* functions shall test whether *c* is a character of class **punct** in the
38709 CX current locale of the process, or in the locale represented by *locale*, respectively; see XBD
38710 Chapter 7 (on page 135).38711 The *c* argument is an **int**, the value of which the application shall ensure is a character
38712 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
38713 any other value, the behavior is undefined.38714 **RETURN VALUE**38715 CX The *ispunct()* and *ispunct_l()* functions shall return non-zero if *c* is a punctuation character;
38716 otherwise, they shall return 0.38717 **ERRORS**38718 The *ispunct_l()* function may fail if:38719 CX [EINVAL] *locale* is not a valid locale object handle.38720 **EXAMPLES**

38721 None.

38722 **APPLICATION USAGE**38723 To ensure applications portability, especially across natural languages, only these functions and
38724 the functions in the reference pages listed in the SEE ALSO section should be used for character
38725 classification.38726 **RATIONALE**

38727 None.

38728 **FUTURE DIRECTIONS**

38729 None.

38730 **SEE ALSO**38731 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *isspace()*, *isupper()*,
38732 *isxdigit()*, *setlocale()*, *uselocale()*

38733 XBD Chapter 7 (on page 135), <ctype.h>, <locale.h>

38734 **CHANGE HISTORY**

38735 First released in Issue 1. Derived from Issue 1 of the SVID.

38736 **Issue 6**

38737 The normative text is updated to avoid use of the term “must” for application requirements.

38738 **Issue 7**

38739 The *ispunct_l()* function is added from The Open Group Technical Standard, 2006, Extended API
38740 Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

isspace()

System Interfaces

38741 **NAME**

38742 isspace, isspace_l — test for a white-space character

38743 **SYNOPSIS**

38744 #include <ctype.h>

38745 int isspace(int c);

38746 CX int isspace_l(int c, locale_t locale);

38747 **DESCRIPTION**

38748 CX For *isspace()*: The functionality described on this reference page is aligned with the ISO C
 38749 standard. Any conflict between the requirements described here and the ISO C standard is
 38750 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

38751 CX The *isspace()* and *isspace_l()* functions shall test whether *c* is a character of class **space** in the
 38752 CX current locale of the process, or in the locale represented by *locale*, respectively; see XBD
 38753 Chapter 7 (on page 135).

38754 The *c* argument is an **int**, the value of which the application shall ensure is a character
 38755 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 38756 any other value, the behavior is undefined.

38757 **RETURN VALUE**

38758 CX The *isspace()* and *isspace_l()* functions shall return non-zero if *c* is a white-space character;
 38759 otherwise, they shall return 0.

38760 **ERRORS**38761 The *isspace_l()* function may fail if:

38762 CX [EINVAL] *locale* is not a valid locale object handle.

38763 **EXAMPLES**

38764 None.

38765 **APPLICATION USAGE**

38766 To ensure applications portability, especially across natural languages, only these functions and
 38767 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38768 classification.

38769 **RATIONALE**

38770 None.

38771 **FUTURE DIRECTIONS**

38772 None.

38773 **SEE ALSO**

38774 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isupper()*,
 38775 *isxdigit()*, *setlocale()*, *uselocale()*

38776 XBD Chapter 7 (on page 135), <ctype.h>, <locale.h>

38777 **CHANGE HISTORY**

38778 First released in Issue 1. Derived from Issue 1 of the SVID.

38779 **Issue 6**

38780 The normative text is updated to avoid use of the term “must” for application requirements.

38781 **Issue 7**

38782 The *isspace_l()* function is added from The Open Group Technical Standard, 2006, Extended API
38783 Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

isunordered()

System Interfaces

38784 **NAME**

38785 isunordered — test if arguments are unordered

38786 **SYNOPSIS**

38787 #include <math.h>

38788 int isunordered(real-floating *x*, real-floating *y*);38789 **DESCRIPTION**

38790 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 38791 conflict between the requirements described here and the ISO C standard is unintentional. This
 38792 volume of POSIX.1-2008 defers to the ISO C standard.

38793 The *isunordered()* macro shall determine whether its arguments are unordered.38794 **RETURN VALUE**

38795 Upon successful completion, the *isunordered()* macro shall return 1 if its arguments are
 38796 unordered, and 0 otherwise.

38797 If *x* or *y* is NaN, 1 shall be returned.38798 **ERRORS**

38799 No errors are defined.

38800 **EXAMPLES**

38801 None.

38802 **APPLICATION USAGE**

38803 The relational and equality operators support the usual mathematical relationships between
 38804 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
 38805 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
 38806 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
 38807 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
 38808 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
 38809 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
 38810 indicates that the argument shall be an expression of **real-floating** type.

38811 **RATIONALE**

38812 None.

38813 **FUTURE DIRECTIONS**

38814 None.

38815 **SEE ALSO**38816 *isgreater()*, *isgreaterequal()*, *isless()*, *islessequal()*, *islessgreater()*

38817 XBD <math.h>

38818 **CHANGE HISTORY**

38819 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38820 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/50 is applied, correcting the RETURN
 38821 VALUE section when *x* or *y* is NaN.

NAME

isupper, isupper_l — test for an uppercase letter

SYNOPSIS

```
#include <ctype.h>

int isupper(int c);
int isupper_l(int c, locale_t locale);
```

DESCRIPTION

For *isupper()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *isupper()* and *isupper_l()* functions shall test whether *c* is a character of class **upper** in the current locale of the process, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isupper()* and *isupper_l()* functions shall return non-zero if *c* is an uppercase letter; otherwise, they shall return 0.

ERRORS

The *isupper_l()* function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum(), *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isxdigit()*, *setlocale()*, *uselocale()*

XBD Chapter 7 (on page 135), *<ctype.h>*, *<locale.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

isupper()*System Interfaces*38862 **Issue 7**

38863 The *isupper_l()* function is added from The Open Group Technical Standard, 2006, Extended API
38864 Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

iswalnum()**NAME**

`iswalnum`, `iswalnum_l` — test for an alphanumeric wide-character code

SYNOPSIS

```
#include <wctype.h>
```

```
int iswalnum(wint_t wc);
```

```
CX int iswalnum_l(wint_t wc, locale_t locale);
```

DESCRIPTION

CX For `iswalnum()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

CX The `iswalnum()` and `iswalnum_l()` functions shall test whether `wc` is a wide-character code representing a character of class **alpha** or **digit** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).

The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro `WEOF`. If the argument has any other value, the behavior is undefined.

RETURN VALUE

CX The `iswalnum()` and `iswalnum_l()` functions shall return non-zero if `wc` is an alphanumeric wide-character code; otherwise, they shall return 0.

ERRORS

The `iswalnum_l()` function may fail if:

CX **[EINVAL]** `locale` is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`

XBD Chapter 7 (on page 135), `<locale.h>`, `<stdio.h>`, `<wctype.h>`

CHANGE HISTORY

First released as a World-wide Portability Interface in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

iswalnum()

System Interfaces

38906		
38907		• The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the <code><wctype.h></code> header rather than <code><wchar.h></code>.
38908	Issue 6	
38909		The normative text is updated to avoid use of the term “must” for application requirements.
38910	Issue 7	
38911		The <i>iswalnum_l()</i> function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.
38912		

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

iswalpha, iswalpha_l — test for an alphabetic wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswalpha(wint_t wc);
CX int iswalpha_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswalpha()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *iswalpha()* and *iswalpha_l()* functions shall test whether *wc* is a wide-character code representing a character of class **alpha** in the current locale of the process, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswalpha()* and *iswalpha_l()* functions shall return non-zero if *wc* is an alphabetic wide-character code; otherwise, they shall return 0.

ERRORS

The *iswalpha_l()* function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum(), *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

XBD Chapter 7 (on page 135), *<locale.h>*, *<wctype.h>*

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

iswalpha()

38954		
38955		• The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the <code><wctype.h></code> header rather than <code><wchar.h></code>.
38956	Issue 6	
38957		The normative text is updated to avoid use of the term “must” for application requirements.
38958	Issue 7	
38959		The <i>iswalpha_l()</i> function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.
38960		

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

iswblank()**NAME**

`iswblank`, `iswblank_l` — test for a blank wide-character code

SYNOPSIS

```
#include <wctype.h>
```

```
int iswblank(wint_t wc);
```

```
CX int iswblank_l(wint_t wc, locale_t locale);
```

DESCRIPTION

CX For `iswblank()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

CX The `iswblank()` and `iswblank_l()` functions shall test whether `wc` is a wide-character code representing a character of class **blank** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).

The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro `WEOF`. If the argument has any other value, the behavior is undefined.

RETURN VALUE

CX The `iswblank()` and `iswblank_l()` functions shall return non-zero if `wc` is a blank wide-character code; otherwise, they shall return 0.

ERRORS

The `iswblank_l()` function may fail if:

CX **[EINVAL]** `locale` is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`iswalnum()`, `iswalphabeta()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`

XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

Issue 7

The `iswblank_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswcntrl()

System Interfaces

39002 **NAME**

39003 iswcntrl, iswcntrl_l — test for a control wide-character code

39004 **SYNOPSIS**

39005 #include <wctype.h>

39006 int iswcntrl(wint_t wc);

39007 CX int iswcntrl_l(wint_t wc, locale_t locale);

39008 **DESCRIPTION**39009 CX For *iswcntrl()*: The functionality described on this reference page is aligned with the ISO C
39010 standard. Any conflict between the requirements described here and the ISO C standard is
39011 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39012 CX The *iswcntrl()* and *iswcntrl_l()* functions shall test whether *wc* is a wide-character code
39013 CX representing a character of class **cntrl** in the current locale of the process, or in the locale
39014 represented by *locale*, respectively; see XBD Chapter 7 (on page 135).39015 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
39016 code corresponding to a valid character in the current locale, or equal to the value of the macro
39017 WEOF. If the argument has any other value, the behavior is undefined.39018 **RETURN VALUE**39019 CX The *iswcntrl()* and *iswcntrl_l()* functions shall return non-zero if *wc* is a control wide-character
39020 code; otherwise, they shall return 0.39021 **ERRORS**39022 The *iswcntrl_l()* function may fail if:39023 CX [EINVAL] *locale* is not a valid locale object handle.39024 **EXAMPLES**

39025 None.

39026 **APPLICATION USAGE**39027 To ensure applications portability, especially across natural languages, only these functions and
39028 the functions in the reference pages listed in the SEE ALSO section should be used for character
39029 classification.39030 **RATIONALE**

39031 None.

39032 **FUTURE DIRECTIONS**

39033 None.

39034 **SEE ALSO**39035 *iswalnum()*, *iswalpha()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,
39036 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

39037 XBD Chapter 7 (on page 135), <locale.h>, <wctype.h>

39038 **CHANGE HISTORY**

39039 First released in Issue 4.

39040 **Issue 5**39041 The following change has been made in this version for alignment with
39042 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39043 • The SYNOPSIS has been changed to indicate that this function and associated data types
39044 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39045 **Issue 6**
39046 The normative text is updated to avoid use of the term “must” for application requirements.
- 39047 **Issue 7**
39048 The `iswcntrl_l()` function is added from The Open Group Technical Standard, 2006, Extended
39049 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

iswctype()

System Interfaces

39050 **NAME**39051 `iswctype, iswctype_l` — test character for a specified class39052 **SYNOPSIS**

```

39053     #include <wctype.h>

39054     int iswctype(wint_t wc, wctype_t charclass);
39055 CX     int iswctype_l(wint_t wc, wctype_t charclass,
39056                     locale_t locale);

```

39057 **DESCRIPTION**

39058 CX For `iswctype()`: The functionality described on this reference page is aligned with the ISO C
 39059 standard. Any conflict between the requirements described here and the ISO C standard is
 39060 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

39061 CX The `iswctype()` and `iswctype_l()` functions shall determine whether the wide-character code `wc`
 39062 CX has the character class `charclass`, returning true or false. The `iswctype()` and `iswctype_l()`
 39063 functions are defined on WEOF and wide-character codes corresponding to the valid character
 39064 CX encodings in the current locale, or in the locale represented by `locale`, respectively. If the `wc`
 39065 argument is not in the domain of the function, the result is undefined. If the value of `charclass` is
 39066 invalid (that is, not obtained by a call to `wctype()` or `charclass` is invalidated by a subsequent call
 39067 to `setlocale()` that has affected category `LC_CTYPE`) the result is unspecified.

39068 **RETURN VALUE**

39069 CX The `iswctype()` and `iswctype_l()` functions shall return non-zero (true) if and only if `wc` has the
 39070 CX property described by `charclass`. If `charclass` is 0, these functions shall return 0.

39071 **ERRORS**

39072 The `iswctype_l()` function may fail if:

39073 CX [EINVAL] `locale` is not a valid locale object handle.

39074 **EXAMPLES**39075 **Testing for a Valid Character**

```

39076     #include <wctype.h>
39077     ...
39078     int yes_or_no;
39079     wint_t wc;
39080     wctype_t valid_class;
39081     ...
39082     if ((valid_class=wctype("vowel")) == (wctype_t)0)
39083         * Invalid character class. */
39084     yes_or_no=iswctype(wc, valid_class);

```

39085 **APPLICATION USAGE**

39086 The twelve strings "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower",
 39087 "print", "punct", "space", "upper", and "xdigit" are reserved for the standard
 39088 character classes. In the table below, the functions in the left column are equivalent to the
 39089 functions in the right column.

39090	<code>iswalnum(wc)</code>	<code>iswctype(wc, wctype("alnum"))</code>
39091	<code>iswalnum_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("alnum"), locale)</code>
39092	<code>iswalpha(wc)</code>	<code>iswctype(wc, wctype("alpha"))</code>
39093	<code>iswalpha_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("alpha"), locale)</code>
39094	<code>iswblank(wc)</code>	<code>iswctype(wc, wctype("blank"))</code>

System Interfaces

iswctype()

39095	<code>iswblank_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("blank"), locale)</code>
39096	<code>iswcntrl(wc)</code>	<code>iswctype(wc, wctype("cntrl"))</code>
39097	<code>iswcntrl_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("cntrl"), locale)</code>
39098	<code>iswdigit(wc)</code>	<code>iswctype(wc, wctype("digit"))</code>
39099	<code>iswdigit_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("digit"), locale)</code>
39100	<code>iswgraph(wc)</code>	<code>iswctype(wc, wctype("graph"))</code>
39101	<code>iswgraph_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("graph"), locale)</code>
39102	<code>iswlower(wc)</code>	<code>iswctype(wc, wctype("lower"))</code>
39103	<code>iswlower_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("lower"), locale)</code>
39104	<code>iswprint(wc)</code>	<code>iswctype(wc, wctype("print"))</code>
39105	<code>iswprint_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("print"), locale)</code>
39106	<code>iswpunct(wc)</code>	<code>iswctype(wc, wctype("punct"))</code>
39107	<code>iswpunct_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("punct"), locale)</code>
39108	<code>iswspace(wc)</code>	<code>iswctype(wc, wctype("space"))</code>
39109	<code>iswspace_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("space"), locale)</code>
39110	<code>iswupper(wc)</code>	<code>iswctype(wc, wctype("upper"))</code>
39111	<code>iswupper_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("upper"), locale)</code>
39112	<code>iswxdigit(wc)</code>	<code>iswctype(wc, wctype("xdigit"))</code>
39113	<code>iswxdigit_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("xdigit"), locale)</code>

39114 **RATIONALE**

39115 None.

39116 **FUTURE DIRECTIONS**

39117 None.

39118 **SEE ALSO**

39119 *iswalnum(), iswalpalpha(), iswcntrl(), iswdigit(), iswgraph(), iswlower(), iswprint(), iswpunct(),*
 39120 *iswspace(), iswupper(), iswxdigit(), setlocale(), uselocale(), wctype()*

39121 XBD **<locale.h>**, **<wctype.h>**39122 **CHANGE HISTORY**

39123 First released as World-wide Portability Interfaces in Issue 4.

39124 **Issue 5**

39125 The following change has been made in this version for alignment with
 39126 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39127 • The SYNOPSIS has been changed to indicate that this function and associated data types
 39128 are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

39129 **Issue 6**39130 The behavior of $n=0$ is now described.

39131 An example is added.

39132 A new function, *iswblank()*, is added to the list in the APPLICATION USAGE.39133 **Issue 7**

39134 The *iswctype_l()* function is added from The Open Group Technical Standard, 2006, Extended
 39135 API Set Part 4.

iswdigit()

System Interfaces

39136 **NAME**39137 `iswdigit, iswdigit_l` — test for a decimal digit wide-character code39138 **SYNOPSIS**39139 `#include <wctype.h>`39140 `int iswdigit(wint_t wc);`39141 CX `int iswdigit_l(wint_t wc, locale_t locale);`39142 **DESCRIPTION**39143 CX For `iswdigit()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39146 CX The `iswdigit()` and `iswdigit_l()` functions shall test whether `wc` is a wide-character code representing a character of class **digit** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39149 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39152 **RETURN VALUE**39153 CX The `iswdigit()` and `iswdigit_l()` functions shall return non-zero if `wc` is a decimal digit wide-character code; otherwise, they shall return 0.39155 **ERRORS**39156 The `iswdigit_l()` function may fail if:39157 CX `[EINVAL]` `locale` is not a valid locale object handle.39158 **EXAMPLES**

39159 None.

39160 **APPLICATION USAGE**

39161 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39164 **RATIONALE**

39165 None.

39166 **FUTURE DIRECTIONS**

39167 None.

39168 **SEE ALSO**39169 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`39171 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39172 **CHANGE HISTORY**

39173 First released in Issue 4.

39174 **Issue 5**39175 The following change has been made in this version for alignment with
39176 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39177 • The SYNOPSIS has been changed to indicate that this function and associated data types
39178 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39179 **Issue 6**
39180 The normative text is updated to avoid use of the term “must” for application requirements.
- 39181 **Issue 7**
39182 The *iswdigit_l()* function is added from The Open Group Technical Standard, 2006, Extended
39183 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

iswgraph()

System Interfaces

39184 **NAME**39185 `iswgraph, iswgraph_l` — test for a visible wide-character code39186 **SYNOPSIS**39187 `#include <wctype.h>`39188 `int iswgraph(wint_t wc);`39189 CX `int iswgraph_l(wint_t wc, locale_t locale);`39190 **DESCRIPTION**39191 CX For `iswgraph()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39194 CX The `iswgraph()` and `iswgraph_l()` functions shall test whether `wc` is a wide-character code representing a character of class **graph** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39197 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39200 **RETURN VALUE**39201 CX The `iswgraph()` and `iswgraph_l()` functions shall return non-zero if `wc` is a wide-character code with a visible representation; otherwise, they shall return 0.39203 **ERRORS**39204 The `iswgraph_l()` function may fail if:39205 CX `[EINVAL]` `locale` is not a valid locale object handle.39206 **EXAMPLES**

39207 None.

39208 **APPLICATION USAGE**

39209 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39212 **RATIONALE**

39213 None.

39214 **FUTURE DIRECTIONS**

39215 None.

39216 **SEE ALSO**39217 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`39219 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39220 **CHANGE HISTORY**

39221 First released in Issue 4.

39222 **Issue 5**39223 The following change has been made in this version for alignment with
39224 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39225 • The SYNOPSIS has been changed to indicate that this function and associated data types
39226 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39227 **Issue 6**
39228 The normative text is updated to avoid use of the term “must” for application requirements.
- 39229 **Issue 7**
39230 The *iswgraph_l()* function is added from The Open Group Technical Standard, 2006, Extended
39231 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

iswlower()

System Interfaces

39232 **NAME**39233 `iswlower, iswlower_l` — test for a lowercase letter wide-character code39234 **SYNOPSIS**39235 `#include <wctype.h>`39236 `int iswlower(wint_t wc);`39237 CX `int iswlower_l(wint_t wc, locale_t locale);`39238 **DESCRIPTION**39239 CX For `iswlower()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39242 CX The `iswlower()` and `iswlower_l()` functions shall test whether `wc` is a wide-character code representing a character of class **lower** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39245 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39248 **RETURN VALUE**39249 CX The `iswlower()` and `iswlower_l()` functions shall return non-zero if `wc` is a lowercase letter wide-character code; otherwise, they shall return 0.39251 **ERRORS**39252 The `iswlower_l()` function may fail if:39253 CX [EINVAL] `locale` is not a valid locale object handle.39254 **EXAMPLES**

39255 None.

39256 **APPLICATION USAGE**

39257 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39260 **RATIONALE**

39261 None.

39262 **FUTURE DIRECTIONS**

39263 None.

39264 **SEE ALSO**39265 `iswalnum()`, `iswalphabeta()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()` (on page 2162) 139267 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39268 **CHANGE HISTORY**

39269 First released in Issue 4.

39270 **Issue 5**39271 The following change has been made in this version for alignment with
39272 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39273 • The SYNOPSIS has been changed to indicate that this function and associated data types
39274 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39275 **Issue 6**
39276 The normative text is updated to avoid use of the term “must” for application requirements.
- 39277 **Issue 7**
39278 The `iswlower_l()` function is added from The Open Group Technical Standard, 2006, Extended
39279 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

iswprint()

System Interfaces

39280 **NAME**39281 `iswprint`, `iswprint_l` — test for a printable wide-character code39282 **SYNOPSIS**39283 `#include <wctype.h>`39284 `int iswprint(wint_t wc);`39285 CX `int iswprint_l(wint_t wc, locale_t locale);`39286 **DESCRIPTION**39287 CX For `iswprint()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39290 CX The `iswprint()` and `iswprint_l()` functions shall test whether `wc` is a wide-character code representing a character of class **print** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39293 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39296 **RETURN VALUE**39297 CX The `iswprint()` and `iswprint_l()` functions shall return non-zero if `wc` is a printable wide-character code; otherwise, they shall return 0.39299 **ERRORS**39300 The `iswprint_l()` function may fail if:39301 CX **[EINVAL]** `locale` is not a valid locale object handle.39302 **EXAMPLES**

39303 None.

39304 **APPLICATION USAGE**

39305 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39308 **RATIONALE**

39309 None.

39310 **FUTURE DIRECTIONS**

39311 None.

39312 **SEE ALSO**39313 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswpunct()`,
39314 `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`39315 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39316 **CHANGE HISTORY**

39317 First released in Issue 4.

39318 **Issue 5**39319 The following change has been made in this version for alignment with
39320 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39321 • The SYNOPSIS has been changed to indicate that this function and associated data types
39322 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39323 **Issue 6**
- 39324 The normative text is updated to avoid use of the term “must” for application requirements.
- 39325 **Issue 7**
- 39326 The *iswprint_l()* function is added from The Open Group Technical Standard, 2006, Extended
39327 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

iswpunct()

System Interfaces

39328 **NAME**39329 `iswpunct, iswpunct_l` — test for a punctuation wide-character code39330 **SYNOPSIS**39331 `#include <wctype.h>`39332 `int iswpunct(wint_t wc);`39333 CX `int iswpunct_l(wint_t wc, locale_t locale);`39334 **DESCRIPTION**39335 CX For `iswpunct()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39338 CX The `iswpunct()` and `iswpunct_l()` functions shall test whether `wc` is a wide-character code representing a character of class **punct** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39341 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39344 **RETURN VALUE**39345 CX The `iswpunct()` and `iswpunct_l()` functions shall return non-zero if `wc` is a punctuation wide-character code; otherwise, they shall return 0.39347 **ERRORS**39348 The `iswpunct_l()` function may fail if:39349 CX [EINVAL] `locale` is not a valid locale object handle.39350 **EXAMPLES**

39351 None.

39352 **APPLICATION USAGE**

39353 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39356 **RATIONALE**

39357 None.

39358 **FUTURE DIRECTIONS**

39359 None.

39360 **SEE ALSO**39361 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`39363 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39364 **CHANGE HISTORY**

39365 First released in Issue 4.

39366 **Issue 5**39367 The following change has been made in this version for alignment with
39368 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39369 • The SYNOPSIS has been changed to indicate that this function and associated data types
39370 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39371 **Issue 6**
39372 The normative text is updated to avoid use of the term “must” for application requirements.
- 39373 **Issue 7**
39374 The `iswpunct_l()` function is added from The Open Group Technical Standard, 2006, Extended
39375 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

iswspace()

System Interfaces

39376 **NAME**39377 `iswspace, iswspace_l` — test for a white-space wide-character code39378 **SYNOPSIS**39379 `#include <wctype.h>`39380 `int iswspace(wint_t wc);`39381 CX `int iswspace_l(wint_t wc, locale_t locale);`39382 **DESCRIPTION**39383 CX For `iswspace()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.39386 CX The `iswspace()` and `iswspace_l()` functions shall test whether `wc` is a wide-character code representing a character of class **space** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).39389 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.39392 **RETURN VALUE**39393 CX The `iswspace()` and `iswspace_l()` functions shall return non-zero if `wc` is a white-space wide-character code; otherwise, they shall return 0.39395 **ERRORS**39396 The `iswspace_l()` function may fail if:39397 CX `[EINVAL]` `locale` is not a valid locale object handle.39398 **EXAMPLES**

39399 None.

39400 **APPLICATION USAGE**

39401 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39404 **RATIONALE**

39405 None.

39406 **FUTURE DIRECTIONS**

39407 None.

39408 **SEE ALSO**39409 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`39411 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`39412 **CHANGE HISTORY**

39413 First released in Issue 4.

39414 **Issue 5**39415 The following change has been made in this version for alignment with
39416 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39417 • The SYNOPSIS has been changed to indicate that this function and associated data types
39418 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39419 **Issue 6**
39420 The normative text is updated to avoid use of the term “must” for application requirements.
- 39421 **Issue 7**
39422 The `iswspace_l()` function is added from The Open Group Technical Standard, 2006, Extended
39423 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

iswupper()

System Interfaces

39424 **NAME**39425 `iswupper, iswupper_l` — test for an uppercase letter wide-character code39426 **SYNOPSIS**

```
39427     #include <wctype.h>
39428     int iswupper(wint_t wc);
39429 CX    int iswupper_l(wint_t wc, locale_t locale);
```

39430 **DESCRIPTION**

39431 CX For `iswupper()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

39434 CX The `iswupper()` and `iswupper_l()` functions shall test whether `wc` is a wide-character code representing a character of class **upper** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).

39437 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

39440 **RETURN VALUE**

39441 CX The `iswupper()` and `iswupper_l()` functions shall return non-zero if `wc` is an uppercase letter wide-character code; otherwise, they shall return 0.

39443 **ERRORS**

39444 The `iswupper_l()` function may fail if:

39445 CX **[EINVAL]** `locale` is not a valid locale object handle.

39446 **EXAMPLES**

39447 None.

39448 **APPLICATION USAGE**

39449 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39452 **RATIONALE**

39453 None.

39454 **FUTURE DIRECTIONS**

39455 None.

39456 **SEE ALSO**

39457 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswxdigit()`, `setlocale()`, `uselocale()`

39459 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`

39460 **CHANGE HISTORY**

39461 First released in Issue 4.

39462 **Issue 5**

39463 The following change has been made in this version for alignment with
39464 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39465 • The SYNOPSIS has been changed to indicate that this function and associated data types
39466 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

39467 **Issue 6**

39468 The normative text is updated to avoid use of the term “must” for application requirements.

39469 **Issue 7**

39470 The `iswupper_l()` function is added from The Open Group Technical Standard, 2006, Extended
39471 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

iswxdigit()

System Interfaces

39472 **NAME**39473 `iswxdigit`, `iswxdigit_l` — test for a hexadecimal digit wide-character code39474 **SYNOPSIS**

```
39475 #include <wctype.h>
39476
39476 int iswxdigit(wint_t wc);
39477 CX int iswxdigit_l(wint_t wc, locale_t locale);
```

39478 **DESCRIPTION**

39479 CX For `iswxdigit()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

39482 CX The `iswxdigit()` and `iswxdigit_l()` functions shall test whether `wc` is a wide-character code representing a character of class **xdigit** in the current locale of the process, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).

39485 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro `WEOF`. If the argument has any other value, the behavior is undefined.

39488 **RETURN VALUE**

39489 CX The `iswxdigit()` and `iswxdigit_l()` functions shall return non-zero if `wc` is a hexadecimal digit wide-character code; otherwise, they shall return 0.

39491 **ERRORS**

39492 The `iswxdigit_l()` function may fail if:

39493 CX **[EINVAL]** `locale` is not a valid locale object handle.

39494 **EXAMPLES**

39495 None.

39496 **APPLICATION USAGE**

39497 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39500 **RATIONALE**

39501 None.

39502 **FUTURE DIRECTIONS**

39503 None.

39504 **SEE ALSO**

39505 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `setlocale()`, `uselocale()`

39507 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`

39508 **CHANGE HISTORY**

39509 First released in Issue 4.

39510 **Issue 5**

39511 The following change has been made in this version for alignment with
39512 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 39513 • The SYNOPSIS has been changed to indicate that this function and associated data types
39514 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.
- 39515 **Issue 6**
- 39516 The normative text is updated to avoid use of the term “must” for application requirements.
- 39517 **Issue 7**
- 39518 The `iswxdigit_l()` function is added from The Open Group Technical Standard, 2006, Extended
39519 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

isxdigit()

System Interfaces

39520 **NAME**

39521 isxdigit, isxdigit_l — test for a hexadecimal digit

39522 **SYNOPSIS**

39523 #include <ctype.h>

39524 int isxdigit(int c);

39525 CX int isxdigit_l(int c, locale_t locale);

39526 **DESCRIPTION**

39527 CX For *isxdigit()*: The functionality described on this reference page is aligned with the ISO C
 39528 standard. Any conflict between the requirements described here and the ISO C standard is
 39529 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

39530 CX The *isxdigit()* and *isxdigit_l()* functions shall test whether *c* is a character of class **xdigit** in the
 39531 CX current locale of the process, or in the locale represented by *locale*, respectively; see XBD
 39532 Chapter 7 (on page 135).

39533 The *c* argument is an **int**, the value of which the application shall ensure is a character
 39534 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 39535 any other value, the behavior is undefined.

39536 **RETURN VALUE**

39537 CX The *isxdigit()* and *isxdigit_l()* functions shall return non-zero if *c* is a hexadecimal digit;
 39538 otherwise, they shall return 0.

39539 **ERRORS**39540 The *isxdigit_l()* function may fail if:

39541 CX [EINVAL] *locale* is not a valid locale object handle.

39542 **EXAMPLES**

39543 None.

39544 **APPLICATION USAGE**

39545 To ensure applications portability, especially across natural languages, only these functions and
 39546 the functions in the reference pages listed in the SEE ALSO section should be used for character
 39547 classification.

39548 **RATIONALE**

39549 None.

39550 **FUTURE DIRECTIONS**

39551 None.

39552 **SEE ALSO**

39553 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*,
 39554 *isupper()*

39555 XBD Chapter 7 (on page 135), <ctype.h>

39556 **CHANGE HISTORY**

39557 First released in Issue 1. Derived from Issue 1 of the SVID.

39558 **Issue 6**

39559 The normative text is updated to avoid use of the term “must” for application requirements.

39560 **Issue 7**

39561 The *isxdigit_l()* function is added from The Open Group Technical Standard, 2006, Extended API
39562 Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

j0()39563 **NAME**39564 `j0, j1, jn` — Bessel functions of the first kind39565 **SYNOPSIS**

```

39566 XSI      #include <math.h>
39567
39567         double j0(double x);
39568         double j1(double x);
39569         double jn(int n, double x);

```

39570 **DESCRIPTION**

39571 The `j0()`, `j1()`, and `jn()` functions shall compute Bessel functions of x of the first kind of orders 0,
 39572 1, and n , respectively.

39573 An application wishing to check for error situations should set `errno` to zero and call
 39574 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or
 39575 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-
 39576 zero, an error has occurred.

39577 **RETURN VALUE**

39578 Upon successful completion, these functions shall return the relevant Bessel value of x of the
 39579 first kind.

39580 If the x argument is too large in magnitude, or the correct result would cause underflow, 0 shall
 39581 be returned and a range error may occur.

39582 If x is NaN, a NaN shall be returned.

39583 **ERRORS**

39584 These functions may fail if:

39585 **Range Error** The value of x was too large in magnitude, or an underflow occurred.

39586 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 39587 then `errno` shall be set to [ERANGE]. If the integer expression
 39588 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the underflow
 39589 floating point exception shall be raised.

39590 No other errors shall occur.

39591 **EXAMPLES**

39592 None.

39593 **APPLICATION USAGE**

39594 On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling &`
 39595 `MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.

39596 **RATIONALE**

39597 None.

39598 **FUTURE DIRECTIONS**

39599 None.

39600 **SEE ALSO**

39601 `feclearexcept()`, `fetestexcept()`, `isnan()`, `y0()`

39602 XBD Section 4.19 (on page 116), `<math.h>`

39603 **CHANGE HISTORY**

39604 First released in Issue 1. Derived from Issue 1 of the SVID.

39605 **Issue 5**39606 The DESCRIPTION is updated to indicate how an application should check for an error. This
39607 text was previously published in the APPLICATION USAGE section.39608 **Issue 6**

39609 The may fail [EDOM] error is removed for the case for NaN.

39610 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling
39611 with the ISO/IEC 9899:1999 standard.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

jrand48()*System Interfaces*39612 **NAME**

39613 jrand48 — generate a uniformly distributed pseudo-random long signed integer

39614 **SYNOPSIS**

```
39615 XSI      #include <stdlib.h>  
39616          long jrand48(unsigned short xsubi[3]);
```

39617 **DESCRIPTION**39618 Refer to *drand48()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

kill()**NAME**

kill — send a signal to a process or a group of processes

SYNOPSIS

```
#include <signal.h>

int kill(pid_t pid, int sig);
```

DESCRIPTION

The `kill()` function shall send a signal to a process or a group of processes specified by `pid`. The signal to be sent is specified by `sig` and is either one from the list given in `<signal.h>` or 0. If `sig` is 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can be used to check the validity of `pid`.

For a process to have permission to send a signal to a process designated by `pid`, unless the sending process has appropriate privileges, the real or effective user ID of the sending process shall match the real or saved set-user-ID of the receiving process.

If `pid` is greater than 0, `sig` shall be sent to the process whose process ID is equal to `pid`.

If `pid` is 0, `sig` shall be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the process group ID of the sender, and for which the process has permission to send a signal.

If `pid` is -1, `sig` shall be sent to all processes (excluding an unspecified set of system processes) for which the process has permission to send that signal.

If `pid` is negative, but not -1, `sig` shall be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the absolute value of `pid`, and for which the process has permission to send a signal.

If the value of `pid` causes `sig` to be generated for the sending process, and if `sig` is not blocked for the calling thread and if no other thread has `sig` unblocked or is waiting in a `sigwait()` function for `sig`, either `sig` or at least one pending unblocked signal shall be delivered to the sending thread before `kill()` returns.

The user ID tests described above shall not be applied when sending SIGCONT to a process that is a member of the same session as the sending process.

An implementation that provides extended security controls may impose further implementation-defined restrictions on the sending of signals, including the null signal. In particular, the system may deny the existence of some or all of the processes specified by `pid`.

The `kill()` function is successful if the process has permission to send `sig` to any of the processes specified by `pid`. If `kill()` fails, no signal shall be sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and `errno` set to indicate the error.

ERRORS

The `kill()` function shall fail if:

[EINVAL]	The value of the <code>sig</code> argument is an invalid or unsupported signal number.
[EPERM]	The process does not have permission to send the signal to any receiving process.

kill()*System Interfaces*

39660 [ESRCH] No process or process group can be found corresponding to that specified by
 39661 *pid*.

39662 **EXAMPLES**

39663 None.

39664 **APPLICATION USAGE**

39665 None.

39666 **RATIONALE**

39667 The semantics for permission checking for *kill()* differed between System V and most other
 39668 implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of
 39669 POSIX.1-2008 agree with System V. Specifically, a set-user-ID process cannot protect itself
 39670 against signals (or at least not against SIGKILL) unless it changes its real user ID. This choice
 39671 allows the user who starts an application to send it signals even if it changes its effective user ID.
 39672 The other semantics give more power to an application that wants to protect itself from the user
 39673 who ran it.

39674 Some implementations provide semantic extensions to the *kill()* function when the absolute
 39675 value of *pid* is greater than some maximum, or otherwise special, value. Negative values are a
 39676 flag to *kill()*. Since most implementations return [ESRCH] in this case, this behavior is not
 39677 included in this volume of POSIX.1-2008, although a conforming implementation could provide
 39678 such an extension.

39679 The unspecified processes to which a signal cannot be sent may include the scheduler or *init*.

39680 There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the
 39681 calling process and that signal is not blocked, that signal would be delivered before *kill()*
 39682 returns. This would permit a process to call *kill()* and be guaranteed that the call never return.
 39683 However, historical implementations that provide only the *signal()* function make only the
 39684 weaker guarantee in this volume of POSIX.1-2008, because they only deliver one signal each
 39685 time a process enters the kernel. Modifications to such implementations to support the
 39686 *sigaction()* function generally require entry to the kernel following return from a signal-catching
 39687 function, in order to restore the signal mask. Such modifications have the effect of satisfying the
 39688 stronger requirement, at least when *sigaction()* is used, but not necessarily when *signal()* is used.
 39689 The standard developers considered making the stronger requirement except when *signal()* is
 39690 used, but felt this would be unnecessarily complex. Implementors are encouraged to meet the
 39691 stronger requirement whenever possible. In practice, the weaker requirement is the same, except
 39692 in the rare case when two signals arrive during a very short window. This reasoning also applies
 39693 to a similar requirement for *sigprocmask()*.

39694 In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID
 39695 security checks. This allows a job control shell to continue a job even if processes in the job have
 39696 altered their user IDs (as in the *su* command). In keeping with the addition of the concept of
 39697 sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any
 39698 process in the same session regardless of user ID security checks. This is less restrictive than BSD
 39699 in the sense that ancestor processes (in the same session) can now be the recipient. It is more
 39700 restrictive than BSD in the sense that descendant processes that form new sessions are now
 39701 subject to the user ID checks. A similar relaxation of security is not necessary for the other job
 39702 control signals since those signals are typically sent by the terminal driver in recognition of
 39703 special characters being typed; the terminal driver bypasses all security checks.

39704 In secure implementations, a process may be restricted from sending a signal to a process having
 39705 a different security label. In order to prevent the existence or nonexistence of a process from
 39706 being used as a covert channel, such processes should appear nonexistent to the sender; that is,
 39707 [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to such processes.

Existing implementations vary on the result of a *kill()* with *pid* indicating an inactive process (a terminated process that has not been waited for by its parent). Some indicate success on such a call (subject to permission checking), while others give an error of [ESRCH]. Since the definition of process lifetime in this volume of POSIX.1-2008 covers inactive processes, the [ESRCH] error as described is inappropriate in this case. In particular, this means that an application cannot have a parent process check for termination of a particular child with *kill()*. (Usually this is done with the null signal; this can be done reliably with *waitpid()*.)

There is some belief that the name *kill()* is misleading, since the function is not always intended to cause process termination. However, the name is common to all historical implementations, and any change would be in conflict with the goal of minimal changes to existing application code.

FUTURE DIRECTIONS

None.

SEE ALSO

getpid(), *raise()*, *setsid()*, *sigaction()*, *sigqueue()*, *wait()*

XBD **<signal.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-user-ID of the calling process is checked in place of its effective user ID. This is a FIPS requirement.
- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The behavior when *pid* is *-1* is now specified. It was previously explicitly unspecified in the POSIX.1-1988 standard.

The normative text is updated to avoid use of the term “must” for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/51 is applied, correcting the RATIONALE section.

killpg()

System Interfaces

39743 **NAME**39744 **killpg** — send a signal to a process group39745 **SYNOPSIS**

```
39746 XSI #include <signal.h>
39747 int killpg(pid_t pgrp, int sig);
```

39748 **DESCRIPTION**39749 The *killpg()* function shall send the signal specified by *sig* to the process group specified by *pgrp*.

39750 If *pgrp* is greater than 1, *killpg(pgrp, sig)* shall be equivalent to *kill(-pgrp, sig)*. If *pgrp* is less than or
 39751 equal to 1, the behavior of *killpg()* is undefined.

39752 **RETURN VALUE**39753 Refer to *kill()*.39754 **ERRORS**39755 Refer to *kill()*.39756 **EXAMPLES**39757 **Sending a Signal to All Other Members of a Process Group**

39758 The following example shows how the calling process could send a signal to all other members
 39759 of its process group. To prevent itself from receiving the signal it first makes itself immune to the
 39760 signal by ignoring it.

```
39761 #include <signal.h>
39762 #include <unistd.h>
39763 ...
39764     if (signal(SIGUSR1, SIG_IGN) == SIG_ERR)
39765         /* Handle error */
39766     if (killpg(getpgrp(), SIGUSR1) == -1)
39767         /* Handle error */;
```

39768 **APPLICATION USAGE**

39769 None.

39770 **RATIONALE**

39771 None.

39772 **FUTURE DIRECTIONS**

39773 None.

39774 **SEE ALSO**39775 *getpgid()*, *getpid()*, *kill()*, *raise()*39776 XBD **<signal.h>**39777 **CHANGE HISTORY**

39778 First released in Issue 4, Version 2.

39779 **Issue 5**

39780 Moved from X/OPEN UNIX extension to BASE.

Issue 6

39781 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/52 is applied, adding the example to the
39782 EXAMPLES section.
39783

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

l64a()*System Interfaces*39784 **NAME**

39785 l64a — convert a 32-bit integer to a radix-64 ASCII string

39786 **SYNOPSIS**

```
39787 XSI      #include <stdlib.h>  
39788          char *l64a(long value);
```

39789 **DESCRIPTION**39790 Refer to *a64l()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

labs()39791 **NAME**

39792 labs, llabs — return a long integer absolute value

39793 **SYNOPSIS**

39794 #include <stdlib.h>

39795 long labs(long i);

39796 long long llabs(long long i);

39797 **DESCRIPTION**

39798 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 39799 conflict between the requirements described here and the ISO C standard is unintentional. This
 39800 volume of POSIX.1-2008 defers to the ISO C standard.

39801 The *labs()* function shall compute the absolute value of the **long** integer operand *i*. The *llabs()*
 39802 function shall compute the absolute value of the **long long** integer operand *i*. If the result
 39803 cannot be represented, the behavior is undefined.

39804 **RETURN VALUE**39805 The *labs()* function shall return the absolute value of the **long** integer operand.39806 The *llabs()* function shall return the absolute value of the **long long** integer operand.39807 **ERRORS**

39808 No errors are defined.

39809 **EXAMPLES**

39810 None.

39811 **APPLICATION USAGE**

39812 None.

39813 **RATIONALE**

39814 None.

39815 **FUTURE DIRECTIONS**

39816 None.

39817 **SEE ALSO**39818 *abs()*

39819 XBD <stdlib.h>

39820 **CHANGE HISTORY**

39821 First released in Issue 4. Derived from the ISO C standard.

39822 **Issue 6**39823 The *llabs()* function is added for alignment with the ISO/IEC 9899:1999 standard.39824 **Issue 7**

39825 SD5-XSH-ERN-152 is applied, correcting the RETURN VALUE section.

lchown()*System Interfaces*39826 **NAME**39827 **lchown** — change the owner and group of a symbolic link39828 **SYNOPSIS**39829 `#include <unistd.h>`39830 `int lchown(const char *path, uid_t owner, gid_t group);`39831 **DESCRIPTION**

39832 The *lchown()* function shall be equivalent to *chown()*, except in the case where the named file is a
 39833 symbolic link. In this case, *lchown()* shall change the ownership of the symbolic link file itself,
 39834 while *chown()* changes the ownership of the file or directory to which the symbolic link refers.

39835 **RETURN VALUE**

39836 Upon successful completion, *lchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to
 39837 indicate an error.

39838 **ERRORS**39839 The *lchown()* function shall fail if:39840 [EACCES] Search permission is denied on a component of the path prefix of *path*.

39841 [EINVAL] The owner or group ID is not a value supported by the implementation.

39842 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 39843 argument.

39844 [ENAMETOOLONG]

39845 The length of a component of a pathname is longer than {NAME_MAX}.

39846 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

39847 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument
 39848 contains at least one non-*<slash>* character and ends with one or more trailing
 39849 *<slash>* characters and the last pathname component names an existing file
 39850 that is neither a directory nor a symbolic link to a directory.

39851 [EPERM] The effective user ID does not match the owner of the file and the process does
 39852 not have appropriate privileges.

39853 [EROFS] The file resides on a read-only file system.

39854 The *lchown()* function may fail if:

39855 [EIO] An I/O error occurred while reading or writing to the file system.

39856 [EINTR] A signal was caught during execution of the function.

39857 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 39858 resolution of the *path* argument.

39859 [ENAMETOOLONG]

39860 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 39861 symbolic link produced an intermediate result with a length that exceeds
 39862 {PATH_MAX}.

EXAMPLES**Changing the Current Owner of a File**

The following example shows how to change the ownership of the symbolic link named **/modules/pass1** to the user ID associated with “jones” and the group ID associated with “cnd”.

The numeric value for the user ID is obtained by using the *getpwnam()* function. The numeric value for the group ID is obtained by using the *getgrnam()* function.

```
#include <sys/types.h>
#include <unistd.h>
#include <pwd.h>
#include <grp.h>

struct passwd *pwd;
struct group *grp;
char          *path = "/modules/pass1";
...
pwd = getpwnam("jones");
grp = getgrnam("cnd");
lchown(path, pwd->pw_uid, grp->gr_gid);
```

APPLICATION USAGE

On implementations which support symbolic links as directory entries rather than files, *lchown()* may fail.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chown(), *symlink()*

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

The Open Group Base Resolution bwg2001-013 is applied, adding wording to the APPLICATION USAGE.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The *lchown()* function is moved from the XSI option to the Base.

The [EOPNOTSUPP] error is removed.

lchown()*System Interfaces*39903
39904

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***lcong48()**39905 **NAME**

39906 lcong48 — seed a uniformly distributed pseudo-random signed long integer generator

39907 **SYNOPSIS**

```
39908 XSI      #include <stdlib.h>  
39909          void lcong48(unsigned short param[7]);
```

39910 **DESCRIPTION**39911 Refer to *drand48()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

ldexp()*System Interfaces*39912 **NAME**

39913 ldexp, ldexpf, ldexpl — load exponent of a floating-point number

39914 **SYNOPSIS**

39915 #include <math.h>

39916 double ldexp(double x, int exp);

39917 float ldexpf(float x, int exp);

39918 long double ldexpl(long double x, int exp);

39919 **DESCRIPTION**

39920 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 39921 conflict between the requirements described here and the ISO C standard is unintentional. This
 39922 volume of POSIX.1-2008 defers to the ISO C standard.

39923 These functions shall compute the quantity $x * 2^{exp}$.

39924 An application wishing to check for error situations should set *errno* to zero and call
 39925 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 39926 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 39927 zero, an error has occurred.

39928 **RETURN VALUE**39929 Upon successful completion, these functions shall return *x* multiplied by 2, raised to the power
39930 *exp*.

39931 If these functions would cause overflow, a range error shall occur and *ldexp()*, *ldexpf()*, and
 39932 *ldexpl()* shall return $\pm$ HUGE_VAL, $\pm$ HUGE_VALF, and $\pm$ HUGE_VALL (according to the sign of
 39933 *x*), respectively.

39934 If the correct value would cause underflow, and is not representable, a range error may occur,
 39935 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

39936 MX If *x* is NaN, a NaN shall be returned.39937 If *x* is ± 0 or $\pm$ Inf, *x* shall be returned.39938 If *exp* is 0, *x* shall be returned.

39939 If the correct value would cause underflow, and is representable, a range error may occur and
 39940 the correct value shall be returned.

39941 **ERRORS**

39942 These functions shall fail if:

39943 Range Error The result overflows.

39944 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 39945 then *errno* shall be set to [ERANGE]. If the integer expression
 39946 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 39947 floating-point exception shall be raised.

39948 These functions may fail if:

39949 Range Error The result underflows.

39950 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 39951 then *errno* shall be set to [ERANGE]. If the integer expression
 39952 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 39953 floating-point exception shall be raised.

39954 **EXAMPLES**

39955 None.

39956 **APPLICATION USAGE**

39957 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 39958 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

39959 **RATIONALE**

39960 None.

39961 **FUTURE DIRECTIONS**

39962 None.

39963 **SEE ALSO**39964 *feclearexcept()*, *fetestexcept()*, *frexp()*, *isnan()*

39965 XBD Section 4.19 (on page 116), <math.h>

39966 **CHANGE HISTORY**

39967 First released in Issue 1. Derived from Issue 1 of the SVID.

39968 **Issue 5**

39969 The DESCRIPTION is updated to indicate how an application should check for an error. This
 39970 text was previously published in the APPLICATION USAGE section.

39971 **Issue 6**

39972 The *ldexpf()* and *ldexpl()* functions are added for alignment with the ISO/IEC 9899:1999
 39973 standard.

39974 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 39975 revised to align with the ISO/IEC 9899:1999 standard.

39976 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 39977 marked.

ldiv()*System Interfaces*39978 **NAME**39979 **ldiv, lldiv** — compute quotient and remainder of a long division39980 **SYNOPSIS**39981 `#include <stdlib.h>`39982 `ldiv_t ldiv(long numer, long denom);`39983 `lldiv_t lldiv(long long numer, long long denom);`39984 **DESCRIPTION**

39985 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 39986 conflict between the requirements described here and the ISO C standard is unintentional. This
 39987 volume of POSIX.1-2008 defers to the ISO C standard.

39988 These functions shall compute the quotient and remainder of the division of the numerator
 39989 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the **long**
 39990 integer (for the *ldiv()* function) or **long long** integer (for the *lldiv()* function) of lesser magnitude
 39991 that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is
 39992 undefined; otherwise, *quot* * *denom* + *rem* shall equal *numer*.

39993 **RETURN VALUE**

39994 The *ldiv()* function shall return a structure of type **ldiv_t**, comprising both the quotient and the
 39995 remainder. The structure shall include the following members, in any order:

39996 `long quot; /* Quotient */`39997 `long rem; /* Remainder */`

39998 The *lldiv()* function shall return a structure of type **lldiv_t**, comprising both the quotient and the
 39999 remainder. The structure shall include the following members, in any order:

40000 `long long quot; /* Quotient */`40001 `long long rem; /* Remainder */`40002 **ERRORS**

40003 No errors are defined.

40004 **EXAMPLES**

40005 None.

40006 **APPLICATION USAGE**

40007 None.

40008 **RATIONALE**

40009 None.

40010 **FUTURE DIRECTIONS**

40011 None.

40012 **SEE ALSO**40013 [*div\(\)*](#)40014 XBD [`<stdlib.h>`](#)40015 **CHANGE HISTORY**

40016 First released in Issue 4. Derived from the ISO C standard.

40017 **Issue 6**40018 The *lldiv()* function is added for alignment with the ISO/IEC 9899:1999 standard.

*System Interfaces***lfind()**40019 **NAME**

40020 lfind — find entry in a linear search table

40021 **SYNOPSIS**

```
40022 XSI #include <search.h>
40023 void *lfind(const void *key, const void *base, size_t *nelp,
40024            size_t width, int (*compar)(const void *, const void *));
```

40025 **DESCRIPTION**40026 Refer to *lsearch()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

lgamma()*System Interfaces*40027 **NAME**

40028 lgamma, lgammaf, lgammal, signgam — log gamma function

40029 **SYNOPSIS**

```

40030 #include <math.h>
40031 double lgamma(double x);
40032 float lgammaf(float x);
40033 long double lgammal(long double x);
40034 XSI extern int signgam;

```

40035 **DESCRIPTION**

40036 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 40037 conflict between the requirements described here and the ISO C standard is unintentional. This
 40038 volume of POSIX.1-2008 defers to the ISO C standard.

40039 These functions shall compute $\log_e |\Gamma(x)|$ where $\Gamma(x)$ is defined as $\int_0^\infty e^{-t} t^{x-1} dt$. The argument x
 40040 need not be a non-positive integer ($\Gamma(x)$ is defined over the reals, except the non-positive
 40041 integers).

40042 XSI If x is NaN, $-\text{Inf}$, or a negative integer, the value of *signgam* is unspecified.

40043 CX These functions need not be thread-safe.

40044 An application wishing to check for error situations should set *errno* to zero and call
 40045 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 40046 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 40047 zero, an error has occurred.

40048 **RETURN VALUE**

40049 Upon successful completion, these functions shall return the logarithmic gamma of x .

40050 If x is a non-positive integer, a pole error shall occur and *lgamma*(), *lgammaf*(), and *lgammal*()
 40051 shall return $+\text{HUGE\_VAL}$, $+\text{HUGE\_VALF}$, and $+\text{HUGE\_VALL}$, respectively.

40052 If the correct value would cause overflow, a range error shall occur and *lgamma*(), *lgammaf*(),
 40053 and *lgammal*() shall return $\pm\text{HUGE\_VAL}$, $\pm\text{HUGE\_VALF}$, and $\pm\text{HUGE\_VALL}$ (having the same
 40054 sign as the correct value), respectively.

40055 MX If x is NaN, a NaN shall be returned.

40056 If x is 1 or 2, $+0$ shall be returned.

40057 If x is $+\text{Inf}$, $+\text{Inf}$ shall be returned.

40058 **ERRORS**

40059 These functions shall fail if:

40060 Pole Error The x argument is a negative integer or zero.

40061 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40062 then *errno* shall be set to [ERANGE]. If the integer expression
 40063 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 40064 floating-point exception shall be raised.

40065 Range Error The result overflows.

40066 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40067 then *errno* shall be set to [ERANGE]. If the integer expression

40068 *(math_errhandling & MATH_ERREXCEPT)* is non-zero, then the overflow
 40069 floating-point exception shall be raised.

EXAMPLES

40070
 40071 None.

APPLICATION USAGE

40072 On error, the expressions *(math_errhandling & MATH_ERRNO)* and *(math_errhandling &*
 40073 *MATH_ERREXCEPT)* are independent of each other, but at least one of them must be non-zero.

RATIONALE

40075
 40076 None.

FUTURE DIRECTIONS

40077
 40078 None.

SEE ALSO

40080 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

40081 XBD Section 4.19 (on page 116), **<math.h>**

CHANGE HISTORY

40082 First released in Issue 3.

Issue 5

40085 The DESCRIPTION is updated to indicate how an application should check for an error. This
 40086 text was previously published in the APPLICATION USAGE section.

40087 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 6

40088 The *lgamma()* function is no longer marked as an extension.

40090 The *lgammaf()* and *lgammal()* functions are added for alignment with the ISO/IEC 9899:1999
 40091 standard.

40092 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 40093 revised to align with the ISO/IEC 9899:1999 standard.

40094 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 40095 marked.

40096 Functionality relating to the XSI option is marked.

Issue 7

40097 Austin Group Interpretation 1003.1-2001 #156 is applied.

40099 The DESCRIPTION is clarified regarding the value of *signgam* when *x* is Nan, *-Inf*, or a negative
 40100 integer.

link()*System Interfaces*40101 **NAME**

40102 link, linkat — link one file to another file relative to two directory file descriptors

40103 **SYNOPSIS**

```
40104 #include <unistd.h>
40105 int link(const char *path1, const char *path2);
40106 int linkat(int fd1, const char *path1, int fd2, const char *path2,
40107           int flag);
```

40108 **DESCRIPTION**40109 The *link()* function shall create a new link (directory entry) for the existing file, *path1*.

40110 The *path1* argument points to a pathname naming an existing file. The *path2* argument points to a pathname naming the new directory entry to be created. The *link()* function shall atomically create a new link for the existing file and the link count of the file shall be incremented by one.

40113 If *path1* names a directory, *link()* shall fail unless the process has appropriate privileges and the implementation supports using *link()* on directories.

40115 If *path1* names a symbolic link, it is implementation-defined whether *link()* follows the symbolic link, or creates a new link to the symbolic link itself.

40117 Upon successful completion, *link()* shall mark for update the last file status change timestamp of the file. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

40120 If *link()* fails, no link shall be created and the link count of the file shall remain unchanged.

40121 The implementation may require that the calling process has permission to access the existing file.

40123 The *linkat()* function shall be equivalent to the *link()* function except in the case where either *path1* or *path2* or both are relative paths. In this case a relative path *path1* is interpreted relative to the directory associated with the file descriptor *fd1* instead of the current working directory and similarly for *path2* and the file descriptor *fd2*. If the file descriptor was opened without *O_SEARCH*, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with *O_SEARCH*, the function shall not perform the check.

40130 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in *<fcntl.h>*:

40132 *AT_SYMLINK_FOLLOW*40133 If *path1* names a symbolic link, a new link for the target of the symbolic link is created.

40134 If *linkat()* is passed the special value *AT_FDCWD* in the *fd1* or *fd2* parameter, the current working directory is used for the respective *path* argument. If both *fd1* and *fd2* have value *AT_FDCWD*, the behavior shall be identical to a call to *link()*.

40137 If the *AT_SYMLINK_FOLLOW* flag is clear in the *flag* argument and the *path1* argument names a symbolic link, a new link is created for the symbolic link *path1* and not its target.

40139 **RETURN VALUE**

40140 Upon successful completion, these functions shall return 0. Otherwise, these functions shall return -1 and set *errno* to indicate the error.

40142 **ERRORS**

40143 These functions shall fail if:

40144 [EACCES] A component of either path prefix denies search permission, or the requested
 40145 link requires writing in a directory that denies write permission, or the calling
 40146 process does not have permission to access the existing file and this is required
 40147 by the implementation.

40148 [EEXIST] The *path2* argument resolves to an existing directory entry or refers to a
 40149 symbolic link.

40150 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path1* or
 40151 *path2* argument.

40152 [EMLINK] The number of links to the file named by *path1* would exceed {LINK_MAX}.

40153 [ENAMETOOLONG]

40154 The length of a component of a pathname is longer than {NAME_MAX}.

40155 [ENOENT] A component of either path prefix does not exist; the file named by *path1* does
 40156 not exist; or *path1* or *path2* points to an empty string.

40157 [ENOSPC] The directory to contain the link cannot be extended.

40158 [ENOTDIR] A component of either path prefix is not a directory, or the *path1* argument
 40159 contains at least one non-`<slash>` character and ends with one or more trailing
 40160 `<slash>` characters and the last pathname component names an existing file
 40161 that is neither a directory nor a symbolic link to a directory.

40162 [EPERM] The file named by *path1* is a directory and either the calling process does not
 40163 have appropriate privileges or the implementation prohibits using *link()* on
 40164 directories.

40165 [EROFS] The requested link requires writing in a directory on a read-only file system.

40166 [EXDEV] The link named by *path2* and the file named by *path1* are on different file
 40167 systems and the implementation does not support links between file systems.

40168 OB XSR [EXDEV] *path1* refers to a named STREAM.

40169 The *linkat()* function shall fail if:

40170 [EBADF] The *path1* or *path2* argument does not specify an absolute path and the *fd1* or
 40171 *fd2* argument, respectively, is neither AT_FDCWD nor a valid file descriptor
 40172 open for reading.

40173 These functions may fail if:

40174 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 40175 resolution of the *path1* or *path2* argument.

40176 [ENAMETOOLONG]

40177 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 40178 symbolic link produced an intermediate result with a length that exceeds
 40179 {PATH_MAX}.

link()*System Interfaces*

40180 The *linkat()* function may fail if:

40181 [EINVAL] The value of the *flag* argument is not valid.

40182 [ENOTDIR] The *path1* or *path2* argument is not an absolute path and *fd1* or *fd2*,
40183 respectively, is neither AT_FDCWD nor a file descriptor associated with a
40184 directory.

40185 **EXAMPLES**40186 **Creating a Link to a File**

40187 The following example shows how to create a link to a file named **/home/cnd/mod1** by creating
40188 a new directory entry named **/modules/pass1**.

```
40189 #include <unistd.h>
40190 char *path1 = "/home/cnd/mod1";
40191 char *path2 = "/modules/pass1";
40192 int status;
40193 ...
40194 status = link (path1, path2);
```

40195 **Creating a Link to a File Within a Program**

40196 In the following program example, the *link()* function links the **/etc/passwd** file (defined as
40197 **PASSWDFILE**) to a file named **/etc/opasswd** (defined as **SAVEFILE**), which is used to save the
40198 current password file. Then, after removing the current password file (defined as
40199 **PASSWDFILE**), the new password file is saved as the current password file using the *link()*
40200 function again.

```
40201 #include <unistd.h>
40202 #define LOCKFILE "/etc/ptmp"
40203 #define PASSWDFILE "/etc/passwd"
40204 #define SAVEFILE "/etc/opasswd"
40205 ...
40206 /* Save current password file */
40207 link (PASSWDFILE, SAVEFILE);
40208
40209 /* Remove current password file. */
40210 unlink (PASSWDFILE);
40211
40212 /* Save new password file as current password file. */
40213 link (LOCKFILE, PASSWDFILE);
```

40212 **APPLICATION USAGE**

40213 Some implementations do allow links between file systems.

40214 If *path1* refers to a symbolic link, application developers should use *linkat()* with appropriate
40215 flags to select whether or not the symbolic link should be resolved.

40216 **RATIONALE**

40217 Linking to a directory is restricted to the superuser in most historical implementations because
40218 this capability may produce loops in the file hierarchy or otherwise corrupt the file system. This
40219 volume of POSIX.1-2008 continues that philosophy by prohibiting *link()* and *unlink()* from
40220 doing this. Other functions could do it if the implementor designed such an extension.

40221 Some historical implementations allow linking of files on different file systems. Wording was

added to explicitly allow this optional behavior.

The exception for cross-file system links is intended to apply only to links that are programmatically indistinguishable from “hard” links.

The purpose of the *linkat()* function is to link files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *link()*, resulting in unspecified behavior. By opening a file descriptor for the directory of both the existing file and the target location and using the *linkat()* function it can be guaranteed that the both filenames are in the desired directories.

The AT_SYMLINK_FOLLOW flag allows for implementing both common behaviors of the *link()* function. The POSIX specification requires that if *path1* is a symbolic link, a new link for the target of the symbolic link is created. Many systems by default or as an alternative provide a mechanism to avoid the implicit symbolic link lookup and create a new link for the symbolic link itself.

Earlier versions of this standard specified only the *link()* function, and required it to behave like *linkat()* with the AT_SYMLINK_FOLLOW flag. However, historical practice from SVR4 and Linux kernels had *link()* behaving like *linkat()* with no flags, and many systems that attempted to provide a conforming *link()* function did so in a way that was rarely used, and when it was used did not conform to the standard (e.g., by not being atomic, or by dereferencing the symbolic link incorrectly). Since applications could not rely on *link()* following links in practice, the *linkat()* function was added taking a flag to specify the desired behavior for the application.

FUTURE DIRECTIONS

None.

SEE ALSO

rename(), *symlink()*, *unlink()*

XBD *<fcntl.h>*, *<unistd.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- An explanation is added of the action when *path2* refers to a symbolic link.
- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

SD5-XSH-ERN-93 is applied, adding RATIONALE.

The *linkat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to XSI STREAMS is marked obsolescent.

Changes are made related to support for finegrained timestamps.

link()

System Interfaces

40264

The [EOPNOTSUPP] error is removed.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

lio_listio()**NAME**

lio_listio — list directed I/O

SYNOPSIS

```
#include <aio.h>
```

```
int lio_listio(int mode, struct aiocb *restrict const list[restrict],
               int nent, struct sigevent *restrict sig);
```

DESCRIPTION

The *lio_listio()* function shall initiate a list of I/O requests with a single function call.

The *mode* argument takes one of the values LIO_WAIT or LIO_NOWAIT declared in *<aio.h>* and determines whether the function returns when the I/O operations have been completed, or as soon as the operations have been queued. If the *mode* argument is LIO_WAIT, the function shall wait until all I/O is complete and the *sig* argument shall be ignored.

If the *mode* argument is LIO_NOWAIT, the function shall return immediately, and asynchronous notification shall occur, according to the *sig* argument, when all the I/O operations complete. If *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous notification occurs as specified in [Section 2.4.1](#) (on page 484) when all the requests in *list* have completed.

The I/O requests enumerated by *list* are submitted in an unspecified order.

The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements. The array may contain NULL elements, which shall be ignored.

If the buffer pointed to by *list* or the **aiocb** structures pointed to by the elements of the array *list* become illegal addresses before all asynchronous I/O completed and, if necessary, the notification is sent, then the behavior is undefined. If the buffers pointed to by the *aio_buf* member of the **aiocb** structure pointed to by the elements of the array *list* become illegal addresses prior to the asynchronous I/O associated with that **aiocb** structure being completed, the behavior is undefined.

The *aio_lio_opcode* field of each **aiocb** structure specifies the operation to be performed. The supported operations are LIO_READ, LIO_WRITE, and LIO_NOP; these symbols are defined in *<aio.h>*. The LIO_NOP operation causes the list entry to be ignored. If the *aio_lio_opcode* element is equal to LIO_READ, then an I/O operation is submitted as if by a call to *aio_read()* with the *aio_cbp* equal to the address of the **aiocb** structure. If the *aio_lio_opcode* element is equal to LIO_WRITE, then an I/O operation is submitted as if by a call to *aio_write()* with the *aio_cbp* equal to the address of the **aiocb** structure.

The *aio_fildes* member specifies the file descriptor on which the operation is to be performed.

The *aio_buf* member specifies the address of the buffer to or from which the data is transferred.

The *aio_nbytes* member specifies the number of bytes of data to be transferred.

The members of the **aiocb** structure further describe the I/O operation to be performed, in a manner identical to that of the corresponding **aiocb** structure when used by the *aio_read()* and *aio_write()* functions.

The *nent* argument specifies how many elements are members of the list; that is, the length of the array.

The behavior of this function is altered according to the definitions of synchronized I/O data integrity completion and synchronized I/O file integrity completion if synchronized I/O is enabled on the file associated with *aio_fildes*.

lio_listio()*System Interfaces*

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *aioctx*→*aio_fildes*.

If *sig*→*sigev_notify* is SIGEV_THREAD and *sig*→*sigev_notify_attributes* is a non-null pointer and the block pointed to by this pointer becomes an illegal address prior to all asynchronous I/O being completed, then the behavior is undefined.

RETURN VALUE

If the *mode* argument has the value LIO_NOWAIT, the *lio_listio()* function shall return the value zero if the I/O operations are successfully queued; otherwise, the function shall return the value -1 and set *errno* to indicate the error.

If the *mode* argument has the value LIO_WAIT, the *lio_listio()* function shall return the value zero when all the indicated I/O has completed successfully. Otherwise, *lio_listio()* shall return a value of -1 and set *errno* to indicate the error.

In either case, the return value only indicates the success or failure of the *lio_listio()* call itself, not the status of the individual I/O requests. In some cases one or more of the I/O requests contained in the list may fail. Failure of an individual request does not prevent completion of any other individual request. To determine the outcome of each I/O request, the application shall examine the error status associated with each **aioctx** control block. The error statuses so returned are identical to those returned as the result of an *aio_read()* or *aio_write()* function.

ERRORS

The *lio_listio()* function shall fail if:

40329 40330 40331	[EAGAIN]	The resources necessary to queue all the I/O requests were not available. The application may check the error status for each aioctx to determine the individual request(s) that failed.
40332 40333	[EAGAIN]	The number of entries indicated by <i>nent</i> would cause the system-wide limit {AIO_MAX} to be exceeded.
40334 40335	[EINVAL]	The <i>mode</i> argument is not a proper value, or the value of <i>nent</i> was greater than {AIO_LISTIO_MAX}.
40336 40337 40338 40339 40340 40341 40342	[EINTR]	A signal was delivered while waiting for all I/O requests to complete during an LIO_WAIT operation. Note that, since each I/O operation invoked by <i>lio_listio()</i> may possibly provoke a signal when it completes, this error return may be caused by the completion of one (or more) of the very I/O operations being awaited. Outstanding I/O requests are not canceled, and the application shall examine each list element to determine whether the request was initiated, canceled, or completed.
40343 40344 40345	[EIO]	One or more of the individual I/O operations failed. The application may check the error status for each aioctx structure to determine the individual request(s) that failed.

In addition to the errors returned by the *lio_listio()* function, if the *lio_listio()* function succeeds or fails with errors of [EAGAIN], [EINTR], or [EIO], then some of the I/O specified by the list may have been initiated. If the *lio_listio()* function fails with an error code other than [EAGAIN], [EINTR], or [EIO], no operations from the list shall have been initiated. The I/O operation indicated by each list element can encounter errors specific to the individual read or write function being performed. In this event, the error status for each **aioctx** control block contains the associated error code. The error codes that can be set are the same as would be set by a *read()* or *write()* function, with the following additional error codes possible:

- 40354 [EAGAIN] The requested I/O operation was not queued due to resource limitations.
- 40355 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
40356 *aio_cancel()* request.
- 40357 [EFBIG] The *aioctx->aio_lio_opcode* is LIO_WRITE, the file is a regular file,
40358 *aioctx->aio_nbytes* is greater than 0, and the *aioctx->aio_offset* is greater than or
40359 equal to the offset maximum in the open file description associated with
40360 *aioctx->aio_fildes*.
- 40361 [EINPROGRESS] The requested I/O is in progress.
- 40362 [EOVERFLOW] The *aioctx->aio_lio_opcode* is LIO_READ, the file is a regular file,
40363 *aioctx->aio_nbytes* is greater than 0, and the *aioctx->aio_offset* is before the
40364 end-of-file and is greater than or equal to the offset maximum in the open file
40365 description associated with *aioctx->aio_fildes*.

EXAMPLES

40366 None.

APPLICATION USAGE

40369 None.

RATIONALE

40371 Although it may appear that there are inconsistencies in the specified circumstances for error
40372 codes, the [EIO] error condition applies when any circumstance relating to an individual
40373 operation makes that operation fail. This might be due to a badly formulated request (for
40374 example, the *aio_lio_opcode* field is invalid, and *aio_error()* returns [EINVAL]) or might arise from
40375 application behavior (for example, the file descriptor is closed before the operation is initiated,
40376 and *aio_error()* returns [EBADF]).

40377 The limitation on the set of error codes returned when operations from the list shall have been
40378 initiated enables applications to know when operations have been started and whether
40379 *aio_error()* is valid for a specific operation.

FUTURE DIRECTIONS

40381 None.

SEE ALSO

40383 *aio_read()*, *aio_write()*, *aio_error()*, *aio_return()*, *aio_cancel()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,
40384 *read()*

40385 XBD <*aio.h*>

CHANGE HISTORY

40387 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

40388 Large File Summit extensions are added.

Issue 6

40390 The [ENOSYS] error condition has been removed as stubs need not be provided if an
40391 implementation does not support the Asynchronous Input and Output option.

40392 The *lio_listio()* function is marked as part of the Asynchronous Input and Output option.

40393 The following new requirements on POSIX implementations derive from alignment with the
40394 Single UNIX Specification:

lio_listio()*System Interfaces*

- 40395 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs
 40396 past the offset maximum established in the open file description associated with
 40397 *aiocbp*→*aio_fildes*. This change is to support large files.
- 40398 • The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support
 40399 large files.
- 40400 The normative text is updated to avoid use of the term “must” for application requirements.
- 40401 The **restrict** keyword is added to the *lio_listio()* prototype for alignment with the
 40402 ISO/IEC 9899:1999 standard.
- 40403 **Issue 6**
- 40404 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/53 is applied, adding new text for
 40405 symmetry with the *aio_read()* and *aio_write()* functions to the DESCRIPTION.
- 40406 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/54 is applied, adding text to the
 40407 DESCRIPTION making it explicit that the user is required to keep the structure pointed to by
 40408 *sig*→*sigev_notify_attributes* valid until the last asynchronous operation finished and the
 40409 notification has been sent.
- 40410 **Issue 7**
- 40411 The *lio_listio()* function is moved from the Asynchronous Input and Output option to the Base.

System Interfaces

listen()**NAME**

listen — listen for socket connections and limit the queue of incoming connections

SYNOPSIS

```
#include <sys/socket.h>

int listen(int socket, int backlog);
```

DESCRIPTION

The *listen()* function shall mark a connection-mode socket, specified by the *socket* argument, as accepting connections.

The *backlog* argument provides a hint to the implementation which the implementation shall use to limit the number of outstanding connections in the socket's listen queue. Implementations may impose a limit on *backlog* and silently reduce the specified value. Normally, a larger *backlog* argument value shall result in a larger or equal length of the listen queue. Implementations shall support values of *backlog* up to SOMAXCONN, defined in *<sys/socket.h>*.

The implementation may include incomplete connections in its listen queue. The limits on the number of incomplete connections and completed connections queued may be different.

The implementation may have an upper limit on the length of the listen queue—either global or per accepting socket. If *backlog* exceeds this limit, the length of the listen queue is set to the limit.

If *listen()* is called with a *backlog* argument value that is less than 0, the function behaves as if it had been called with a *backlog* argument value of 0.

A *backlog* argument of 0 may allow the socket to accept connections, in which case the length of the listen queue may be set to an implementation-defined minimum value.

The socket in use may require the process to have appropriate privileges to use the *listen()* function.

RETURN VALUE

Upon successful completions, *listen()* shall return 0; otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *listen()* function shall fail if:

[EBADF] The *socket* argument is not a valid file descriptor.

[EDESTADDRREQ]

The socket is not bound to a local address, and the protocol does not support listening on an unbound socket.

[EINVAL] The *socket* is already connected.

[ENOTSOCK] The *socket* argument does not refer to a socket.

[EOPNOTSUPP] The socket protocol does not support *listen()*.

The *listen()* function may fail if:

[EACCES] The calling process does not have appropriate privileges.

[EINVAL] The *socket* has been shut down.

[ENOBUFS] Insufficient resources are available in the system to complete the call.

listen()*System Interfaces*40451 **EXAMPLES**

40452 None.

40453 **APPLICATION USAGE**

40454 None.

40455 **RATIONALE**

40456 None.

40457 **FUTURE DIRECTIONS**

40458 None.

40459 **SEE ALSO**40460 *accept()*, *connect()*, *socket()*40461 XBD *<sys/socket.h>*40462 **CHANGE HISTORY**

40463 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

40464 The DESCRIPTION is updated to describe the relationship of SOMAXCONN and the *backlog*
40465 argument.

*System Interfaces***llabs()**40466 **NAME**

40467 llabs — return a long integer absolute value

40468 **SYNOPSIS**

40469 #include <stdlib.h>

40470 long long llabs(long long i);

40471 **DESCRIPTION**40472 Refer to *labs()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

lldiv()*System Interfaces*40473 **NAME**40474 **lldiv** — compute quotient and remainder of a long division40475 **SYNOPSIS**40476 `#include <stdlib.h>`40477 `lldiv_t lldiv(long long numer, long long denom);`40478 **DESCRIPTION**40479 Refer to *ldiv()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

llrint, llrintf, llrintl — round to the nearest integer value using current rounding direction

SYNOPSIS

```
#include <math.h>

long long llrint(double x);
long long llrintf(float x);
long long llrintl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding according to the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

MX If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

MX If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

MX If the correct value is positive and too large to represent as a **long long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur;

CX otherwise, a domain error may occur.

MX If the correct value is negative and too large to represent as a **long long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur;

CX otherwise, a domain error may occur.

ERRORS

These functions shall fail if:

MX **Domain Error** The *x* argument is NaN or $\pm\text{Inf}$, or the correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

MX **Domain Error** The correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

lrint()

System Interfaces

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions provide floating-to-integer conversions. They round according to the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified and the invalid floating-point exception is raised. When they raise no other floating-point exception and the result differs from the argument, they raise the inexact floating-point exception.

FUTURE DIRECTIONS

None.

SEE ALSO*feclearexcept()*, *fetestexcept()*, *lrint()*

XBD Section 4.19 (on page 116), <math.h>

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

Issue 7

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #53 is applied.

NAME

llround, llroundf, llroundl — round to nearest integer value

SYNOPSIS

```
#include <math.h>

long long llround(double x);
long long llroundf(float x);
long long llroundl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur;

otherwise, a domain error may occur.

If the correct value is negative and too large to represent as a **long long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur;

otherwise, a domain error may occur.

ERRORS

These functions shall fail if:

Domain Error The *x* argument is NaN or $\pm$ Inf, or the correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error The correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

llround()

System Interfaces

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions differ from the *llrint()* functions in that the default rounding direction for the *llround()* functions round halfway cases away from zero and need not raise the inexact floating-point exception for non-integer arguments that round to within the range of the return type.

FUTURE DIRECTIONS

None.

SEE ALSO*feclearexcept()*, *fetestexcept()*, *lround()*

XBD Section 4.19 (on page 116), <math.h>

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

Issue 7

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-75) is applied.

System Interfaces

localeconv()

NAME

localeconv — return locale-specific information

SYNOPSIS

```
#include <locale.h>
```

```
struct lconv *localeconv(void);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *localeconv()* function shall set the components of an object with the type **struct lconv** with the values appropriate for the formatting of numeric quantities (monetary and otherwise) according to the rules of the current locale.

The members of the structure with type **char *** are pointers to strings, any of which (except **decimal_point**) can point to "", to indicate that the value is not available in the current locale or is of zero length. The members with type **char** are non-negative numbers, any of which can be {CHAR_MAX} to indicate that the value is not available in the current locale.

The members include the following:

char *decimal_point

The radix character used to format non-monetary quantities.

char *thousands_sep

The character used to separate groups of digits before the decimal-point character in formatted non-monetary quantities.

char *grouping

A string whose elements taken as one-byte integer values indicate the size of each group of digits in formatted non-monetary quantities.

char *int_curr_symbol

The international currency symbol applicable to the current locale. The first three characters contain the alphabetic international currency symbol in accordance with those specified in the ISO 4217:2001 standard. The fourth character (immediately preceding the null byte) is the character used to separate the international currency symbol from the monetary quantity.

char *currency_symbol

The local currency symbol applicable to the current locale.

char *mon_decimal_point

The radix character used to format monetary quantities.

char *mon_thousands_sep

The separator for groups of digits before the decimal-point in formatted monetary quantities.

char *mon_grouping

A string whose elements taken as one-byte integer values indicate the size of each group of digits in formatted monetary quantities.

char *positive_sign

The string used to indicate a non-negative valued formatted monetary quantity.

localeconv()*System Interfaces*

40649	char *negative_sign
40650	The string used to indicate a negative valued formatted monetary quantity.
40651	char int_frac_digits
40652	The number of fractional digits (those after the decimal-point) to be displayed in an
40653	internationally formatted monetary quantity.
40654	char frac_digits
40655	The number of fractional digits (those after the decimal-point) to be displayed in a
40656	formatted monetary quantity.
40657	char p_cs_precedes
40658	Set to 1 if the currency_symbol precedes the value for a non-negative formatted monetary
40659	quantity. Set to 0 if the symbol succeeds the value.
40660	char p_sep_by_space
40661	Set to a value indicating the separation of the currency_symbol , the sign string, and the
40662	value for a non-negative formatted monetary quantity.
40663	char n_cs_precedes
40664	Set to 1 if the currency_symbol precedes the value for a negative formatted monetary
40665	quantity. Set to 0 if the symbol succeeds the value.
40666	char n_sep_by_space
40667	Set to a value indicating the separation of the currency_symbol , the sign string, and the
40668	value for a negative formatted monetary quantity.
40669	char p_sign_posn
40670	Set to a value indicating the positioning of the positive_sign for a non-negative formatted
40671	monetary quantity.
40672	char n_sign_posn
40673	Set to a value indicating the positioning of the negative_sign for a negative formatted
40674	monetary quantity.
40675	char int_p_cs_precedes
40676	Set to 1 or 0 if the int_curr_symbol respectively precedes or succeeds the value for a non-
40677	negative internationally formatted monetary quantity.
40678	char int_n_cs_precedes
40679	Set to 1 or 0 if the int_curr_symbol respectively precedes or succeeds the value for a
40680	negative internationally formatted monetary quantity.
40681	char int_p_sep_by_space
40682	Set to a value indicating the separation of the int_curr_symbol , the sign string, and the
40683	value for a non-negative internationally formatted monetary quantity.
40684	char int_n_sep_by_space
40685	Set to a value indicating the separation of the int_curr_symbol , the sign string, and the
40686	value for a negative internationally formatted monetary quantity.
40687	char int_p_sign_posn
40688	Set to a value indicating the positioning of the positive_sign for a non-negative
40689	internationally formatted monetary quantity.
40690	char int_n_sign_posn
40691	Set to a value indicating the positioning of the negative_sign for a negative internationally
40692	formatted monetary quantity.

40693	The elements of grouping and mon_grouping are interpreted according to the following:
40694	{CHAR_MAX} No further grouping is to be performed.
40695	0 The previous element is to be repeatedly used for the remainder of the digits.
40696	<i>other</i> The integer value is the number of digits that comprise the current group. The
40697	next element is examined to determine the size of the next group of digits
40698	before the current group.
40699	The values of p_sep_by_space , n_sep_by_space , int_p_sep_by_space , and int_n_sep_by_space
40700	are interpreted according to the following:
40701	0 No space separates the currency symbol and value.
40702	1 If the currency symbol and sign string are adjacent, a space separates them from the value;
40703	otherwise, a space separates the currency symbol from the value.
40704	2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a
40705	space separates the sign string from the value.
40706	For int_p_sep_by_space and int_n_sep_by_space , the fourth character of int_curr_symbol is
40707	used instead of a space.
40708	The values of p_sign_posn , n_sign_posn , int_p_sign_posn , and int_n_sign_posn are
40709	interpreted according to the following:
40710	0 Parentheses surround the quantity and currency_symbol or int_curr_symbol .
40711	1 The sign string precedes the quantity and currency_symbol or int_curr_symbol .
40712	2 The sign string succeeds the quantity and currency_symbol or int_curr_symbol .
40713	3 The sign string immediately precedes the currency_symbol or int_curr_symbol .
40714	4 The sign string immediately succeeds the currency_symbol or int_curr_symbol .
40715	The implementation shall behave as if no function in this volume of POSIX.1-2008 calls
40716	<i>localeconv</i> ().
40717	CX The <i>localeconv</i> () function need not be thread-safe.
40718	RETURN VALUE
40719	The <i>localeconv</i> () function shall return a pointer to the filled-in object. The application shall not
40720	modify the structure pointed to by the return value which may be overwritten by a subsequent
40721	call to <i>localeconv</i> (). In addition, calls to <i>setlocale</i> () with the categories <i>LC_ALL</i> , <i>LC_MONETARY</i> ,
40722	or <i>LC_NUMERIC</i> or calls to <i>uselocale</i> () which change the categories <i>LC_MONETARY</i> or
40723	<i>LC_NUMERIC</i> may overwrite the contents of the structure.
40724	ERRORS
40725	No errors are defined.

localeconv()*System Interfaces***EXAMPLES**

None.

APPLICATION USAGE

The following table illustrates the rules which may be used by four countries to format monetary quantities.

Country	Positive Format	Negative Format	International Format
Italy	€1.230	–€1.230	EUR.1.230
Netherlands	€ 1.234,56	€ –1.234,56	EUR 1.234,56
Norway	kr1.234,56	kr1.234,56–	NOK 1.234,56
Switzerland	SFrs.1,234.56	SFrs.1,234.56C	CHF 1,234.56

For these four countries, the respective values for the monetary members of the structure returned by *localeconv()* are:

	Italy	Netherlands	Norway	Switzerland
int_curr_symbol	"EUR. "	"EUR "	"NOK "	"CHF "
currency_symbol	"€." "	"€"	"kr"	"SFrs. "
mon_decimal_point	" "	" , "	" , "	" . "
mon_thousands_sep	" . "	" . "	" . "	" , "
mon_grouping	"\3"	"\3"	"\3"	"\3"
positive_sign	" "	" "	" "	" "
negative_sign	" – "	" – "	" – "	" C "
int_frac_digits	0	2	2	2
frac_digits	0	2	2	2
p_cs_precedes	1	1	1	1
p_sep_by_space	0	1	0	0
n_cs_precedes	1	1	1	1
n_sep_by_space	0	1	0	0
p_sign_posn	1	1	1	1
n_sign_posn	1	4	2	2
int_p_cs_precedes	1	1	1	1
int_n_cs_precedes	1	1	1	1
int_p_sep_by_space	0	0	0	0
int_n_sep_by_space	0	0	0	0
int_p_sign_posn	1	1	1	1
int_n_sign_posn	1	4	4	2

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf(), *fscanf()*, *isalpha()*, *isascii()*, *nl_langinfo()*, *setlocale()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*, *strcpy()*, *strftime()*, *strlen()*, *strpbrk()*, *strspn()*, *strtok()*, *strxfrm()*, *strtod()*, *uselocale()*

XBD [<langinfo.h>](#), [<locale.h>](#)

CHANGE HISTORY

First released in Issue 4. Derived from the ANSI C standard.

Issue 6

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

The RETURN VALUE section is rewritten to avoid use of the term “must”.

This reference page is updated for alignment with the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/31 is applied, removing references to **int_curr_symbol** and updating the descriptions of **p_sep_by_space** and **n_sep_by_space**. These changes are for alignment with the ISO C standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

The definitions of **int_curr_symbol** and **currency_symbol** are updated.

The examples in the APPLICATION USAGE section are updated.

localtime()*System Interfaces*40782 **NAME**40783 `localtime, localtime_r` — convert a time value to a broken-down local time40784 **SYNOPSIS**40785 `#include <time.h>`40786 `struct tm *localtime(const time_t *timer);`40787 CX `struct tm *localtime_r(const time_t *restrict timer,`
40788 `struct tm *restrict result);`40789 **DESCRIPTION**40790 CX For `localtime()`: The functionality described on this reference page is aligned with the ISO C
40791 standard. Any conflict between the requirements described here and the ISO C standard is
40792 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.40793 The `localtime()` function shall convert the time in seconds since the Epoch pointed to by `timer`
40794 into a broken-down time, expressed as a local time. The function corrects for the timezone and
40795 CX any seasonal time adjustments. Local timezone information is used as though `localtime()` calls
40796 `tzset()`.40797 The relationship between a time in seconds since the Epoch used as an argument to `localtime()`
40798 and the `tm` structure (defined in the `<time.h>` header) is that the result shall be as specified in
40799 the expression given in the definition of seconds since the Epoch (see XBD Section 4.15, on page
40800 113) corrected for timezone and any seasonal time adjustments, where the names in the structure
40801 and in the expression correspond.40802 The same relationship shall apply for `localtime_r()`.40803 The `localtime()` function need not be thread-safe.40804 The `asctime()`, `ctime()`, `gmtime()`, and `localtime()` functions shall return values in one of two static
40805 objects: a broken-down time structure and an array of type `char`. Execution of any of the
40806 functions may overwrite the information returned in either of these objects by any of the other
40807 functions.40808 The `localtime_r()` function shall convert the time in seconds since the Epoch pointed to by `timer`
40809 into a broken-down time stored in the structure to which `result` points. The `localtime_r()` function
40810 shall also return a pointer to that same structure.40811 Unlike `localtime()`, the `localtime_r()` function is not required to set `tzname`. If `localtime_r()` does
40812 not set `tzname`, it shall not set `daylight` and shall not set `timezone`.40813 **RETURN VALUE**40814 Upon successful completion, the `localtime()` function shall return a pointer to the broken-down
40815 CX time structure. If an error is detected, `localtime()` shall return a null pointer and set `errno` to
40816 indicate the error.40817 Upon successful completion, `localtime_r()` shall return a pointer to the structure pointed to by the
40818 argument `result`. If an error is detected, `localtime_r()` shall return a null pointer and set `errno` to
40819 indicate the error.40820 **ERRORS**40821 CX The `localtime()` and `localtime_r()` functions shall fail if:

40822 CX [EOVERFLOW] The result cannot be represented.

EXAMPLES

Getting the Local Date and Time

The following example uses the *time()* function to calculate the time elapsed, in seconds, since January 1, 1970 0:00 UTC (the Epoch), *localtime()* to convert that value to a broken-down time, and *asctime()* to convert the broken-down time values into a printable string.

```
#include <stdio.h>
#include <time.h>

int main(void)
{
    time_t result;

    result = time(NULL);
    printf("%sju secs since the Epoch\n",
        asctime(localtime(&result)),
        (uintmax_t)result);
    return(0);
}
```

This example writes the current time to *stdout* in a form like this:

```
Wed Jun 26 10:32:15 1996
835810335 secs since the Epoch
```

Getting the Modification Time for a File

The following example prints the last data modification timestamp in the local timezone for a given file.

```
#include <stdio.h>
#include <time.h>
#include <sys/stat.h>

int
print_file_time(const char *filename)
{
    struct stat statbuf;
    struct tm *tm;
    char timestr[BUFSIZ];

    if (stat(filename, &statbuf) == -1)
        return -1;
    if ((tm = localtime(&statbuf.st_mtime)) == NULL)
        return -1;
    if (strftime(timestr, sizeof(timestr), "%Y-%m-%d %H:%M:%S", tm) == 0)
        return -1;
    printf("%s: %.09ld\n", filename, timestr, statbuf.st_mtim.tv_nsec);
    return 0;
}
```

localtime()*System Interfaces***Timing an Event**

The following example gets the current time, converts it to a string using *localtime()* and *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to an event being timed.

```
#include <time.h>
#include <stdio.h>
...
time_t now;
int minutes_to_event;
...
time(&now);
printf("The time is ");
fputs(asctime(localtime(&now)), stdout);
printf("There are still %d minutes to the event.\n",
      minutes_to_event);
...
```

APPLICATION USAGE

The *localtime_r()* function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime(), *clock()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *tzset()*, *utime()*

XBD Section 4.15 (on page 113), [time.h](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A note indicating that the *localtime()* function need not be reentrant is added to the DESCRIPTION.

The *localtime_r()* function is included for alignment with the POSIX Threads Extension.

Issue 6

The *localtime_r()* function is marked as part of the Thread-Safe Functions option.

Extensions beyond the ISO C standard are marked.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *localtime_r()* prototype for alignment with the ISO/IEC 9899:1999 standard.

Examples are added.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/32 is applied, adding the [EOverflow] error.

- 40906 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/55 is applied, updating the error handling
40907 for *localtime_r()*.
- 40908 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/56 is applied, adding a requirement that if
40909 *localtime_r()* does not set the *tzname* variable, it shall not set the *daylight* or *timezone* variables. On
40910 systems supporting XSI, the *daylight*, *timezone*, and *tzname* variables should all be set to provide
40911 information for the same timezone. This updates the description of *localtime_r()* to mention
40912 *daylight* and *timezone* as well as *tzname*. The SEE ALSO section is updated.
- 40913 **Issue 7**
- 40914 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 40915 The *localtime_r()* function is moved from the Thread-Safe Functions option to the Base.
- 40916 Changes are made to the EXAMPLES section related to support for finegrained timestamps.

lockf()

NAME

lockf — record locking on files

SYNOPSIS

```
XSI #include <unistd.h>
int lockf(int fildes, int function, off_t size);
```

DESCRIPTION

The *lockf()* function shall lock sections of a file with advisory-mode locks. Calls to *lockf()* from threads in other processes which attempt to lock the locked file section shall either return an error value or block until the section becomes unlocked. All the locks for a process are removed when the process terminates. Record locking with *lockf()* shall be supported for regular files and may be supported for other files.

The *fildes* argument is an open file descriptor. To establish a lock with this function, the file descriptor shall be opened with write-only permission (O_WRONLY) or with read/write permission (O_RDWR).

The *function* argument is a control value which specifies the action to be taken. The permissible values for *function* are defined in <unistd.h> as follows:

Function	Description
F_ULOCK	Unlock locked sections.
F_LOCK	Lock a section for exclusive use.
F_TLOCK	Test and lock a section for exclusive use.
F_TEST	Test a section for locks by other processes.

F_TEST shall detect if a lock by another process is present on the specified section.

F_LOCK and F_TLOCK shall both lock a section of a file if the section is available.

F_ULOCK shall remove locks from a section of the file.

The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be locked or unlocked starts at the current offset in the file and extends forward for a positive size or backward for a negative size (the preceding bytes up to but not including the current offset). If *size* is 0, the section from the current offset through the largest possible file offset shall be locked (that is, from the current offset through the present or any future end-of-file). An area need not be allocated to the file to be locked because locks may exist past the end-of-file.

The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent locked sections would occur, the sections shall be combined into a single locked section. If the request would cause the number of locks to exceed a system-imposed limit, the request shall fail.

F_LOCK and F_TLOCK requests differ only by the action taken if the section is not available. F_LOCK shall block the calling thread until the section is available. F_TLOCK shall cause the function to fail if the section is already locked by another process.

File locks shall be released on first close by the locking process of any file descriptor for the file.

F_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the process. Locked sections shall be unlocked starting at the current file offset through *size* bytes or to the end-of-file if *size* is (off_t)0. When all of a locked section is not released (that is, when the beginning or end of the area to be unlocked falls within a locked section), the remaining portions of that section shall remain locked by the process. Releasing the center portion of a locked section shall cause the remaining locked beginning and end portions to become two separate

locked sections. If the request would cause the number of locks in the system to exceed a system-imposed limit, the request shall fail.

A potential for deadlock occurs if the threads of a process controlling a locked section are blocked by accessing a locked section of another process. If the system detects that deadlock would occur, *lockf()* shall fail with an [EDEADLK] error.

The interaction between *fcntl()* and *lockf()* locks is unspecified.

Blocking on a section shall be interrupted by any signal.

An F_ULOCK request in which *size* is non-zero and the offset of the last byte of the requested section is the maximum value for an object of type *off_t*, when the process has an existing lock in which *size* is 0 and which includes the last byte of the requested section, shall be treated as a request to unlock from the start of the requested section with a size equal to 0. Otherwise, an F_ULOCK request shall attempt to unlock only the requested section.

Attempting to lock a section of a file that is associated with a buffered stream produces unspecified results.

RETURN VALUE

Upon successful completion, *lockf()* shall return 0. Otherwise, it shall return -1, set *errno* to indicate an error, and existing locks shall not be changed.

ERRORS

The *lockf()* function shall fail if:

[EBADF] The *filides* argument is not a valid open file descriptor; or *function* is F_LOCK or F_TLOCK and *filides* is not a valid file descriptor open for writing.

[EACCES] or [EAGAIN]

The *function* argument is F_TLOCK or F_TEST and the section is already locked by another process.

[EDEADLK] The *function* argument is F_LOCK and a deadlock is detected.

[EINTR] A signal was caught during execution of the function.

[EINVAL] The *function* argument is not one of F_LOCK, F_TLOCK, F_TEST, or F_ULOCK; or *size* plus the current file offset is less than 0.

[EOVERFLOW] The offset of the first, or if *size* is not 0 then the last, byte in the requested section cannot be represented correctly in an object of type *off_t*.

The *lockf()* function may fail if:

[EAGAIN] The *function* argument is F_LOCK or F_TLOCK and the file is mapped with *mmap()*.

[EDEADLK] or [ENOLCK]

The *function* argument is F_LOCK, F_TLOCK, or F_ULOCK, and the request would cause the number of locks to exceed a system-imposed limit.

[EOPNOTSUPP] or [EINVAL]

The implementation does not support the locking of files of the type indicated by the *filides* argument.

lockf()*System Interfaces*41000 **EXAMPLES**41001 **Locking a Portion of a File**

41002 In the following example, a file named **/home/cnd/mod1** is being modified. Other processes that
 41003 use locking are prevented from changing it during this process. Only the first 10 000 bytes are
 41004 locked, and the lock call fails if another process has any part of this area locked already.

```
41005 #include <fcntl.h>
41006 #include <unistd.h>
41007
41008 int fildes;
41009 int status;
41009 ...
41010 fildes = open("/home/cnd/mod1", O_RDWR);
41011 status = lockf(fildes, F_TLOCK, (off_t)10000);
```

41012 **APPLICATION USAGE**

41013 Record-locking should not be used in combination with the *fopen()*, *fread()*, *fwrite()*, and other
 41014 *stdio* functions. Instead, the more primitive, non-buffered functions (such as *open()*) should be
 41015 used. Unexpected results may occur in processes that do buffering in the user address space. The
 41016 process may later read/write data which is/was locked. The *stdio* functions are the most
 41017 common source of unexpected buffering.

41018 The *alarm()* function may be used to provide a timeout facility in applications requiring it.

41019 **RATIONALE**

41020 None.

41021 **FUTURE DIRECTIONS**

41022 None.

41023 **SEE ALSO**

41024 *alarm()*, *chmod()*, *close()*, *creat()*, *fcntl()*, *fopen()*, *mmap()*, *open()*, *read()*, *write()*

41025 XBD **<unistd.h>**

41026 **CHANGE HISTORY**

41027 First released in Issue 4, Version 2.

41028 **Issue 5**

41029 Moved from X/OPEN UNIX extension to BASE.

41030 Large File Summit extensions are added. In particular, the description of [EINVAL] is clarified
 41031 and moved from optional to mandatory status.

41032 A note is added to the DESCRIPTION indicating the effects of attempting to lock a section of a
 41033 file that is associated with a buffered stream.

41034 **Issue 6**

41035 The normative text is updated to avoid use of the term “must” for application requirements.

41036 **Issue 7**

41037 Austin Group Interpretation 1003.1-2001 #054 is applied, updating the DESCRIPTION.

41038 **NAME**

41039 log, logf, logl — natural logarithm function

41040 **SYNOPSIS**

```
41041 #include <math.h>
41042 double log(double x);
41043 float logf(float x);
41044 long double logl(long double x);
```

41045 **DESCRIPTION**

41046 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41047 conflict between the requirements described here and the ISO C standard is unintentional. This
 41048 volume of POSIX.1-2008 defers to the ISO C standard.

41049 These functions shall compute the natural logarithm of their argument x , $\log_e(x)$.

41050 An application wishing to check for error situations should set *errno* to zero and call
 41051 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 41052 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 41053 zero, an error has occurred.

41054 **RETURN VALUE**

41055 Upon successful completion, these functions shall return the natural logarithm of x .

41056 If x is ± 0 , a pole error shall occur and *log()*, *logf()*, and *logl()* shall return $-\text{HUGE\_VAL}$,
 41057 $-\text{HUGE\_VALF}$, and $-\text{HUGE\_VALL}$, respectively.

41058 MX For finite values of x that are less than 0, or if x is $-\text{Inf}$, a domain error shall occur, and either a
 41059 NaN (if supported), or an implementation-defined value shall be returned.

41060 MX If x is NaN, a NaN shall be returned.

41061 If x is 1, +0 shall be returned.

41062 If x is $+\text{Inf}$, x shall be returned.

41063 **ERRORS**

41064 These functions shall fail if:

41065 MX Domain Error The finite value of x is negative, or x is $-\text{Inf}$.

41066 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41067 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 41068 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 41069 shall be raised.

41070 Pole Error The value of x is zero.

41071 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41072 then *errno* shall be set to [ERANGE]. If the integer expression
 41073 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 41074 floating-point exception shall be raised.

log()*System Interfaces*41075 **EXAMPLES**

41076 None.

41077 **APPLICATION USAGE**

41078 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 41079 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

41080 **RATIONALE**

41081 None.

41082 **FUTURE DIRECTIONS**

41083 None.

41084 **SEE ALSO**41085 *exp()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*, *log10()*, *log1p()*

41086 XBD Section 4.19 (on page 116), <math.h>

41087 **CHANGE HISTORY**

41088 First released in Issue 1. Derived from Issue 1 of the SVID.

41089 **Issue 5**

41090 The DESCRIPTION is updated to indicate how an application should check for an error. This
 41091 text was previously published in the APPLICATION USAGE section.

41092 **Issue 6**

41093 The normative text is updated to avoid use of the term “must” for application requirements.

41094 The *logf()* and *logl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

41095 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 41096 revised to align with the ISO/IEC 9899:1999 standard.

41097 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 41098 marked.

41099 NAME

41100 `log10`, `log10f`, `log10l` — base 10 logarithm function

41101 SYNOPSIS

```
41102 #include <math.h>
41103 double log10(double x);
41104 float log10f(float x);
41105 long double log10l(long double x);
```

41106 DESCRIPTION

41107 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41108 conflict between the requirements described here and the ISO C standard is unintentional. This
 41109 volume of POSIX.1-2008 defers to the ISO C standard.

41110 These functions shall compute the base 10 logarithm of their argument x , $\log_{10}(x)$.

41111 An application wishing to check for error situations should set *errno* to zero and call
 41112 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 41113 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 41114 zero, an error has occurred.

41115 RETURN VALUE

41116 Upon successful completion, these functions shall return the base 10 logarithm of x .

41117 If x is ± 0 , a pole error shall occur and *log10()*, *log10f()*, and *log10l()* shall return `-HUGE_VAL`,
 41118 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

41119 MX For finite values of x that are less than 0, or if x is `-Inf`, a domain error shall occur, and either a
 41120 NaN (if supported), or an implementation-defined value shall be returned.

41121 MX If x is NaN, a NaN shall be returned.

41122 If x is 1, `+0` shall be returned.

41123 If x is `+Inf`, `+Inf` shall be returned.

41124 ERRORS

41125 These functions shall fail if:

41126 MX Domain Error The finite value of x is negative, or x is `-Inf`.

41127 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41128 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 41129 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 41130 shall be raised.

41131 Pole Error The value of x is zero.

41132 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41133 then *errno* shall be set to [ERANGE]. If the integer expression
 41134 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 41135 floating-point exception shall be raised.

log10()*System Interfaces*41136 **EXAMPLES**

41137 None.

41138 **APPLICATION USAGE**

41139 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 41140 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

41141 **RATIONALE**

41142 None.

41143 **FUTURE DIRECTIONS**

41144 None.

41145 **SEE ALSO**41146 *feclearexcept()*, *fetestexcept()*, *isnan()*, *log()*, *pow()*

41147 XBD Section 4.19 (on page 116), <math.h>

41148 **CHANGE HISTORY**

41149 First released in Issue 1. Derived from Issue 1 of the SVID.

41150 **Issue 5**

41151 The DESCRIPTION is updated to indicate how an application should check for an error. This
 41152 text was previously published in the APPLICATION USAGE section.

41153 **Issue 6**

41154 The normative text is updated to avoid use of the term “must” for application requirements.

41155 The *log10f()* and *log10l()* functions are added for alignment with the ISO/IEC 9899:1999
 41156 standard.

41157 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 41158 revised to align with the ISO/IEC 9899:1999 standard.

41159 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 41160 marked.

41161 **NAME**

41162 log1p, log1pf, log1pl — compute a natural logarithm

41163 **SYNOPSIS**

```
41164 #include <math.h>
41165 double log1p(double x);
41166 float log1pf(float x);
41167 long double log1pl(long double x);
```

41168 **DESCRIPTION**

41169 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41170 conflict between the requirements described here and the ISO C standard is unintentional. This
 41171 volume of POSIX.1-2008 defers to the ISO C standard.

41172 These functions shall compute $\log_e(1.0 + x)$.

41173 An application wishing to check for error situations should set *errno* to zero and call
 41174 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 41175 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 41176 zero, an error has occurred.

41177 **RETURN VALUE**

41178 Upon successful completion, these functions shall return the natural logarithm of $1.0 + x$.

41179 If x is -1 , a pole error shall occur and *log1p()*, *log1pf()*, and *log1pl()* shall return $-\text{HUGE\_VAL}$,
 41180 $-\text{HUGE\_VALF}$, and $-\text{HUGE\_VALL}$, respectively.

41181 MX For finite values of x that are less than -1 , or if x is $-\text{Inf}$, a domain error shall occur, and either a
 41182 NaN (if supported), or an implementation-defined value shall be returned.

41183 MX If x is NaN, a NaN shall be returned.

41184 If x is ± 0 , or $+\text{Inf}$, x shall be returned.

41185 If x is subnormal, a range error may occur and x should be returned.

41186 **ERRORS**

41187 These functions shall fail if:

41188 MX Domain Error The finite value of x is less than -1 , or x is $-\text{Inf}$.

41189 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41190 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 41191 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 41192 shall be raised.

41193 Pole Error The value of x is -1 .

41194 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41195 then *errno* shall be set to [ERANGE]. If the integer expression
 41196 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 41197 floating-point exception shall be raised.

41198 These functions may fail if:

41199 MX Range Error The value of x is subnormal.

41200 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41201 then *errno* shall be set to [ERANGE]. If the integer expression
 41202 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 41203 floating-point exception shall be raised.

log1p()

System Interfaces

41204 **EXAMPLES**

41205 None.

41206 **APPLICATION USAGE**

41207 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 41208 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

41209 **RATIONALE**

41210 None.

41211 **FUTURE DIRECTIONS**

41212 None.

41213 **SEE ALSO**41214 *feclearexcept()*, *fetestexcept()*, *log()*

41215 XBD Section 4.19 (on page 116), <math.h>

41216 **CHANGE HISTORY**

41217 First released in Issue 4, Version 2.

41218 **Issue 5**

41219 Moved from X/OPEN UNIX extension to BASE.

41220 **Issue 6**

41221 The normative text is updated to avoid use of the term “must” for application requirements.

41222 The *log1p()* function is no longer marked as an extension.

41223 The *log1pf()* and *log1pl()* functions are added for alignment with the ISO/IEC 9899:1999
 41224 standard.

41225 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 41226 revised to align with the ISO/IEC 9899:1999 standard.

41227 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 41228 marked.

41229 NAME

41230 log2, log2f, log2l — compute base 2 logarithm functions

41231 SYNOPSIS

```
41232 #include <math.h>
41233 double log2(double x);
41234 float log2f(float x);
41235 long double log2l(long double x);
```

41236 DESCRIPTION

41237 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41238 conflict between the requirements described here and the ISO C standard is unintentional. This
 41239 volume of POSIX.1-2008 defers to the ISO C standard.

41240 These functions shall compute the base 2 logarithm of their argument x , $\log_2(x)$.

41241 An application wishing to check for error situations should set *errno* to zero and call
 41242 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 41243 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 41244 zero, an error has occurred.

41245 RETURN VALUE

41246 Upon successful completion, these functions shall return the base 2 logarithm of x .

41247 If x is ± 0 , a pole error shall occur and *log2()*, *log2f()*, and *log2l()* shall return $-\text{HUGE\_VAL}$,
 41248 $-\text{HUGE\_VALF}$, and $-\text{HUGE\_VALL}$, respectively.

41249 MX For finite values of x that are less than 0, or if x is $-\text{Inf}$, a domain error shall occur, and either a
 41250 NaN (if supported), or an implementation-defined value shall be returned.

41251 MX If x is NaN, a NaN shall be returned.

41252 If x is 1, +0 shall be returned.

41253 If x is $+\text{Inf}$, x shall be returned.

41254 ERRORS

41255 These functions shall fail if:

41256 MX Domain Error The finite value of x is less than zero, or x is $-\text{Inf}$.

41257 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41258 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 41259 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 41260 shall be raised.

41261 Pole Error The value of x is zero.

41262 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41263 then *errno* shall be set to [ERANGE]. If the integer expression
 41264 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 41265 floating-point exception shall be raised.

log2()*System Interfaces*41266 **EXAMPLES**

41267 None.

41268 **APPLICATION USAGE**

41269 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
41270 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

41271 **RATIONALE**

41272 None.

41273 **FUTURE DIRECTIONS**

41274 None.

41275 **SEE ALSO**41276 *feclearexcept()*, *fetestexcept()*, *log()*41277 XBD [Section 4.19](#) (on page 116), [<math.h>](#)41278 **CHANGE HISTORY**

41279 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

41280 **NAME**

41281 logb, logbf, logbl — radix-independent exponent

41282 **SYNOPSIS**

```
41283 #include <math.h>
41284 double logb(double x);
41285 float logbf(float x);
41286 long double logbl(long double x);
```

41287 **DESCRIPTION**

41288 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41289 conflict between the requirements described here and the ISO C standard is unintentional. This
 41290 volume of POSIX.1-2008 defers to the ISO C standard.

41291 These functions shall compute the exponent of x , which is the integral part of $\log_r |x|$, as a
 41292 signed floating-point value, for non-zero x , where r is the radix of the machine's floating-point
 41293 arithmetic, which is the value of FLT_RADIX defined in the <float.h> header.

41294 If x is subnormal it is treated as though it were normalized; thus for finite positive x :

41295 $1 \leq x * \text{FLT\_RADIX}^{-\text{logb}(x)} < \text{FLT\_RADIX}$

41296 An application wishing to check for error situations should set *errno* to zero and call
 41297 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 41298 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 41299 zero, an error has occurred.

41300 **RETURN VALUE**

41301 Upon successful completion, these functions shall return the exponent of x .

41302 If x is ± 0 , *logb()*, *logbf()*, and *logbl()* shall return *-HUGE_VAL*, *-HUGE_VALF*, and
 41303 MX *-HUGE_VALL*, respectively. On systems that support the IEC 60559 Floating-Point option, a
 41304 pole error shall occur;

41305 CX otherwise, a pole error may occur.

41306 MX If x is NaN, a NaN shall be returned.

41307 MX If x is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned.

41308 **ERRORS**

41309 These functions shall fail if:

41310 MX **Pole Error** The value of x is ± 0 .

41311 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41312 then *errno* shall be set to [ERANGE]. If the integer expression
 41313 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 41314 floating-point exception shall be raised.

41315 These functions may fail if:

41316 **Pole Error** The value of x is 0.

41317 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41318 then *errno* shall be set to [ERANGE]. If the integer expression
 41319 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 41320 floating-point exception shall be raised.

logb()

System Interfaces

41321 **EXAMPLES**

41322 None.

41323 **APPLICATION USAGE**

41324 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 41325 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

41326 **RATIONALE**

41327 None.

41328 **FUTURE DIRECTIONS**

41329 None.

41330 **SEE ALSO**41331 *feclearexcept()*, *fetestexcept()*, *ilogb()*, *scalbln()*41332 XBD Section 4.19 (on page 116), **<float.h>**, **<math.h>**41333 **CHANGE HISTORY**

41334 First released in Issue 4, Version 2.

41335 **Issue 5**

41336 Moved from X/OPEN UNIX extension to BASE.

41337 **Issue 6**41338 The *logb()* function is no longer marked as an extension.41339 The *logbf()* and *logbl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

41340 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 41341 revised to align with the ISO/IEC 9899:1999 standard.

41342 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 41343 marked.

41344 **Issue 7**

41345 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #50 (SD5-XSH-ERN-76) is applied.

*System Interfaces***logf()**41346 **NAME**

41347 logf, logl — natural logarithm function

41348 **SYNOPSIS**

41349 #include <math.h>

41350 float logf(float x);

41351 long double logl(long double x);

41352 **DESCRIPTION**41353 Refer to *log()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

longjmp()

System Interfaces

41354 **NAME**41355 `longjmp` — non-local goto41356 **SYNOPSIS**41357 `#include <setjmp.h>`41358 `void longjmp(jmp_buf env, int val);`41359 **DESCRIPTION**

41360 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41361 conflict between the requirements described here and the ISO C standard is unintentional. This
 41362 volume of POSIX.1-2008 defers to the ISO C standard.

41363 The *longjmp()* function shall restore the environment saved by the most recent invocation of
 41364 *setjmp()* in the same thread, with the corresponding **jmp_buf** argument. If there is no such
 41365 invocation, or if the function containing the invocation of *setjmp()* has terminated execution in
 41366 the interim, or if the invocation of *setjmp()* was within the scope of an identifier with variably
 41367 CX modified type and execution has left that scope in the interim, the behavior is undefined. It is
 41368 unspecified whether *longjmp()* restores the signal mask, leaves the signal mask unchanged, or
 41369 restores it to its value at the time *setjmp()* was called.

41370 All accessible objects have values, and all other components of the abstract machine have state
 41371 (for example, floating-point status flags and open files), as of the time *longjmp()* was called,
 41372 except that the values of objects of automatic storage duration are unspecified if they meet all
 41373 the following conditions:

- 41374 • They are local to the function containing the corresponding *setjmp()* invocation.
- 41375 • They do not have volatile-qualified type.
- 41376 • They are changed between the *setjmp()* invocation and *longjmp()* call.

41377 CX As it bypasses the usual function call and return mechanisms, *longjmp()* shall execute correctly
 41378 in contexts of interrupts, signals, and any of their associated functions. However, if *longjmp()* is
 41379 invoked from a nested signal handler (that is, from a function invoked as a result of a signal
 41380 raised during the handling of another signal), the behavior is undefined.

41381 The effect of a call to *longjmp()* where initialization of the **jmp_buf** structure was not performed
 41382 in the calling thread is undefined.

41383 **RETURN VALUE**

41384 After *longjmp()* is completed, program execution continues as if the corresponding invocation of
 41385 *setjmp()* had just returned the value specified by *val*. The *longjmp()* function shall not cause
 41386 *setjmp()* to return 0; if *val* is 0, *setjmp()* shall return 1.

41387 **ERRORS**

41388 No errors are defined.

41389 **EXAMPLES**

41390 None.

41391 **APPLICATION USAGE**

41392 Applications whose behavior depends on the value of the signal mask should not use *longjmp()*
 41393 and *setjmp()*, since their effect on the signal mask is unspecified, but should instead use the
 41394 *siglongjmp()* and *sigsetjmp()* functions (which can save and restore the signal mask under
 41395 application control).

41396 RATIONALE

41397 None.

41398 FUTURE DIRECTIONS

41399 None.

41400 SEE ALSO

41401 *setjmp()*, *sigaction()*, *siglongjmp()*, *sigsetjmp()*

41402 XBD **<setjmp.h>**

41403 CHANGE HISTORY

41404 First released in Issue 1. Derived from Issue 1 of the SVID.

41405 Issue 5

41406 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

41407 Issue 6

41408 Extensions beyond the ISO C standard are marked.

41409 The following new requirements on POSIX implementations derive from alignment with the
41410 Single UNIX Specification:

- 41411 • The DESCRIPTION now explicitly makes *longjmp()*'s effect on the signal mask
- 41412 unspecified.

41413 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

lrand48()*System Interfaces*41414 **NAME**

41415 lrand48 — generate uniformly distributed pseudo-random non-negative long integers

41416 **SYNOPSIS**

```
41417 XSI #include <stdlib.h>  
41418 long lrand48(void);
```

41419 **DESCRIPTION**41420 Refer to *drand48()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

`lrint`, `lrintf`, `lrintl` — round to nearest integer value using current rounding direction

SYNOPSIS

```
#include <math.h>

long lrint(double x);
long lrintf(float x);
long lrintl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding according to the current rounding direction.

An application wishing to check for error situations should set `errno` to zero and call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

If x is NaN, a domain error shall occur and an unspecified value is returned.

If x is $+\text{Inf}$, a domain error shall occur and an unspecified value is returned.

If x is $-\text{Inf}$, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur;

otherwise, a domain error may occur.

If the correct value is negative and too large to represent as a **long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur;

otherwise, a domain error may occur.

ERRORS

These functions shall fail if:

Domain Error The x argument is NaN or $\pm\text{Inf}$, or the correct value is not representable as an integer.

If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero, then `errno` shall be set to [EDOM]. If the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error The correct value is not representable as an integer.

If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero, then `errno` shall be set to [EDOM]. If the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the invalid floating-point exception shall be raised.

lrint()

System Interfaces

41465 **EXAMPLES**

41466 None.

41467 **APPLICATION USAGE**

41468 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
41469 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

41470 **RATIONALE**

41471 These functions provide floating-to-integer conversions. They round according to the current
41472 rounding direction. If the rounded value is outside the range of the return type, the numeric
41473 result is unspecified and the invalid floating-point exception is raised. When they raise no other
41474 floating-point exception and the result differs from the argument, they raise the inexact floating-
41475 point exception.

41476 **FUTURE DIRECTIONS**

41477 None.

41478 **SEE ALSO**41479 *feclearexcept()*, *fetestexcept()*, *llrint()*

41480 XBD Section 4.19 (on page 116), <math.h>

41481 **CHANGE HISTORY**

41482 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

41483 **Issue 7**

41484 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #53 (SD5-XSH-ERN-77) is applied.

NAME

lround, lroundf, lroundl — round to nearest integer value

SYNOPSIS

```
#include <math.h>

long lround(double x);
long lroundf(float x);
long lroundl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

CX If the correct value is positive and too large to represent as a **long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain shall occur; otherwise, a domain error may occur.

MX If the correct value is negative and too large to represent as a **long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain shall occur; otherwise, a domain error may occur.

ERRORS

These functions shall fail if:

MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error The correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

lround()

System Interfaces

41527 **EXAMPLES**

41528 None.

41529 **APPLICATION USAGE**

41530 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
41531 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

41532 **RATIONALE**

41533 These functions differ from the *lrint()* functions in the default rounding direction, with the
41534 *lround()* functions rounding halfway cases away from zero and needing not to raise the inexact
41535 floating-point exception for non-integer arguments that round to within the range of the return
41536 type.

41537 **FUTURE DIRECTIONS**

41538 None.

41539 **SEE ALSO**41540 *feclearexcept()*, *fetestexcept()*, *llround()*

41541 XBD Section 4.19 (on page 116), <math.h>

41542 **CHANGE HISTORY**

41543 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

41544 **Issue 7**

41545 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-78) is applied.

System Interfaces

lsearch()**NAME**

lsearch, lfind — linear search and update

SYNOPSIS

```

#include <search.h>

void *lsearch(const void *key, void *base, size_t *nel, size_t width,
              int (*compar)(const void *, const void *));
void *lfind(const void *key, const void *base, size_t *nel,
            size_t width, int (*compar)(const void *, const void *));

```

DESCRIPTION

The *lsearch()* function shall linearly search the table and return a pointer into the table for the matching entry. If the entry does not occur, it shall be added at the end of the table. The *key* argument points to the entry to be sought in the table. The *base* argument points to the first element in the table. The *width* argument is the size of an element in bytes. The *nel* argument points to an integer containing the current number of elements in the table. The integer to which *nel* points shall be incremented if the entry is added to the table. The *compar* argument points to a comparison function which the application shall supply (for example, *strcmp()*). It is called with two arguments that point to the elements being compared. The application shall ensure that the function returns 0 if the elements are equal, and non-zero otherwise.

The *lfind()* function shall be equivalent to *lsearch()*, except that if the entry is not found, it is not added to the table. Instead, a null pointer is returned.

RETURN VALUE

If the searched for entry is found, both *lsearch()* and *lfind()* shall return a pointer to it. Otherwise, *lfind()* shall return a null pointer and *lsearch()* shall return a pointer to the newly added element.

Both functions shall return a null pointer in case of error.

ERRORS

No errors are defined.

EXAMPLES**Storing Strings in a Table**

This fragment reads in less than or equal to TABSIZE strings of length less than or equal to ELSIZE and stores them in a table, eliminating duplicates.

```

#include <stdio.h>
#include <string.h>
#include <search.h>

#define TABSIZE 50
#define ELSIZE 120

...
    char line[ELSIZE], tab[TABSIZE][ELSIZE];
    size_t nel = 0;
    ...
    while (fgets(line, ELSIZE, stdin) != NULL && nel < TABSIZE)
        (void) lsearch(line, tab, &nel,
                       ELSIZE, (int (*)(const void *, const void *)) strcmp);
    ...

```

lsearch()*System Interfaces***Finding a Matching Entry**

The following example finds any line that reads "This is a test.".

```
#include <search.h>
#include <string.h>
...
char line[ELSIZE], tab[TABSIZE][ELSIZE];
size_t nel = 0;
char *findline;
void *entry;

findline = "This is a test.\n";

entry = lfind(findline, tab, &nel, ELSIZE, (
    int (*)(const void *, const void *)) strcmp);
```

APPLICATION USAGE

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Undefined results can occur if there is not enough room in the table to add a new item.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

hcreate(), *tdelete()*

XBD [<search.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

NAME

lseek — move the read/write file offset

SYNOPSIS

```
#include <unistd.h>
```

```
off_t lseek(int fildev, off_t offset, int whence);
```

DESCRIPTION

The *lseek()* function shall set the file offset for the open file description associated with the file descriptor *fildev*, as follows:

- If *whence* is SEEK_SET, the file offset shall be set to *offset* bytes.
- If *whence* is SEEK_CUR, the file offset shall be set to its current location plus *offset*.
- If *whence* is SEEK_END, the file offset shall be set to the size of the file plus *offset*.

The symbolic constants SEEK_SET, SEEK_CUR, and SEEK_END are defined in **<unistd.h>**.

The behavior of *lseek()* on devices which are incapable of seeking is implementation-defined. The value of the file offset associated with such a device is undefined.

The *lseek()* function shall allow the file offset to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap shall return bytes with the value 0 until data is actually written into the gap.

The *lseek()* function shall not, by itself, extend the size of a file.

SHM If *fildev* refers to a shared memory object, the result of the *lseek()* function is unspecified.

TYM If *fildev* refers to a typed memory object, the result of the *lseek()* function is unspecified.

RETURN VALUE

Upon successful completion, the resulting offset, as measured in bytes from the beginning of the file, shall be returned. Otherwise, (off_t)−1 shall be returned, *errno* shall be set to indicate the error, and the file offset shall remain unchanged.

ERRORS

The *lseek()* function shall fail if:

[EBADF]	The <i>fildev</i> argument is not an open file descriptor.
[EINVAL]	The <i>whence</i> argument is not a proper value, or the resulting file offset would be negative for a regular file, block special file, or directory.
[EOVERFLOW]	The resulting file offset would be a value which cannot be represented correctly in an object of type off_t .
[ESPIPE]	The <i>fildev</i> argument is associated with a pipe, FIFO, or socket.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The ISO C standard includes the functions *fgetpos()* and *fsetpos()*, which work on very large files by use of a special positioning type.

Although *lseek()* may position the file offset beyond the end of the file, this function does not itself extend the size of the file. While the only function in POSIX.1-2008 that may directly extend

lseek()*System Interfaces*

the size of the file is *write()*, *truncate()*, and *ftruncate()*, several functions originally derived from the ISO C standard, such as *fwrite()*, *fprintf()*, and so on, may do so (by causing calls on *write()*).

An invalid file offset that would cause [EINVAL] to be returned may be both implementation-defined and device-dependent (for example, memory may have few invalid values). A negative file offset may be valid for some devices in some implementations.

The POSIX.1-1990 standard did not specifically prohibit *lseek()* from returning a negative offset. Therefore, an application was required to clear *errno* prior to the call and check *errno* upon return to determine whether a return value of (**off_t**)−1 is a negative offset or an indication of an error condition. The standard developers did not wish to require this action on the part of a conforming application, and chose to require that *errno* be set to [EINVAL] when the resulting file offset would be negative for a regular file, block special file, or directory.

FUTURE DIRECTIONS

None.

SEE ALSO

open()

XBD **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EOVERFLOW] error condition is added. This change is to support large files.

An additional [ESPIPE] error condition is added for sockets.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *lseek()* results are unspecified for typed memory objects.

*System Interfaces***lstat()**41690 **NAME**

41691 lstat — get file status

41692 **SYNOPSIS**

41693 #include <sys/stat.h>

41694 int lstat(const char *restrict *path*, struct stat *restrict *buf*);41695 **DESCRIPTION**41696 Refer to *fstatat()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

malloc()*System Interfaces*41697 **NAME**

41698 malloc — a memory allocator

41699 **SYNOPSIS**

41700 #include <stdlib.h>

41701 void *malloc(size_t size);

41702 **DESCRIPTION**

41703 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41704 conflict between the requirements described here and the ISO C standard is unintentional. This
 41705 volume of POSIX.1-2008 defers to the ISO C standard.

41706 The *malloc()* function shall allocate unused space for an object whose size in bytes is specified by
 41707 *size* and whose value is unspecified.

41708 The order and contiguity of storage allocated by successive calls to *malloc()* is unspecified. The
 41709 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
 41710 a pointer to any type of object and then used to access such an object in the space allocated (until
 41711 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object
 41712 disjoint from any other object. The pointer returned points to the start (lowest byte address) of
 41713 the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of
 41714 the space requested is 0, the behavior is implementation-defined: the value returned shall be
 41715 either a null pointer or a unique pointer.

41716 **RETURN VALUE**

41717 Upon successful completion with *size* not equal to 0, *malloc()* shall return a pointer to the
 41718 allocated space. If *size* is 0, either a null pointer or a unique pointer that can be successfully
 41719 CX passed to *free()* shall be returned. Otherwise, it shall return a null pointer and set *errno* to
 41720 indicate the error.

41721 **ERRORS**41722 The *malloc()* function shall fail if:

41723 CX [ENOMEM] Insufficient storage space is available.

41724 **EXAMPLES**

41725 None.

41726 **APPLICATION USAGE**

41727 None.

41728 **RATIONALE**

41729 None.

41730 **FUTURE DIRECTIONS**

41731 None.

41732 **SEE ALSO**41733 *calloc()*, *free()*, *getrlimit()*, *posix_memalign()*, *realloc()*

41734 XBD <stdlib.h>

41735 **CHANGE HISTORY**

41736 First released in Issue 1. Derived from Issue 1 of the SVID.

41737 **Issue 6**

41738 Extensions beyond the ISO C standard are marked.

- 41739 The following new requirements on POSIX implementations derive from alignment with the
41740 Single UNIX Specification:
- 41741 • In the RETURN VALUE section, the requirement to set *errno* to indicate an error is added.
 - 41742 • The [ENOMEM] error condition is added.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

mblen()*System Interfaces*41743 **NAME**41744 **mblen** — get number of bytes in a character41745 **SYNOPSIS**41746 `#include <stdlib.h>`41747 `int mblen(const char *s, size_t n);`41748 **DESCRIPTION**

41749 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41750 conflict between the requirements described here and the ISO C standard is unintentional. This
 41751 volume of POSIX.1-2008 defers to the ISO C standard.

41752 If *s* is not a null pointer, *mblen()* shall determine the number of bytes constituting the character
 41753 pointed to by *s*. Except that the shift state of *mbtowc()* is not affected, it shall be equivalent to:

41754 `mbtowc((wchar_t *)0, s, n);`

41755 The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls
 41756 *mblen()*.

41757 The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a
 41758 state-dependent encoding, this function shall be placed into its initial state by a call for which its
 41759 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null
 41760 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a
 41761 null pointer shall cause this function to return a non-zero value if encodings have state
 41762 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift
 41763 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an
 41764 adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be
 41765 unspecified.

41766 **RETURN VALUE**

41767 If *s* is a null pointer, *mblen()* shall return a non-zero or 0 value, if character encodings,
 41768 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mblen()* shall
 41769 either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the
 41770 character (if the next *n* or fewer bytes form a valid character), or return -1 (if they do not form a
 41771 CX valid character) and may set *errno* to indicate the error. In no case shall the value returned be
 41772 greater than *n* or the value of the {MB_CUR_MAX} macro.

41773 **ERRORS**41774 The *mblen()* function may fail if:

41775 XSI [EILSEQ] An invalid character sequence is detected.

41776 **EXAMPLES**

41777 None.

41778 **APPLICATION USAGE**

41779 None.

41780 **RATIONALE**

41781 None.

41782 **FUTURE DIRECTIONS**

41783 None.

41784 **SEE ALSO**41785 [mbtowc\(\)](#), [mbstowcs\(\)](#), [wctomb\(\)](#), [wcstombs\(\)](#)41786 XBD [<stdlib.h>](#)41787 **CHANGE HISTORY**

41788 First released in Issue 4. Aligned with the ISO C standard.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

mbrlen()*System Interfaces*41789 **NAME**41790 **mbrlen** — get number of bytes in a character (restartable)41791 **SYNOPSIS**41792 `#include <wchar.h>`

41793 `size_t mbrlen(const char *restrict s, size_t n,`
 41794 `mbstate_t *restrict ps);`

41795 **DESCRIPTION**

41796 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 41797 conflict between the requirements described here and the ISO C standard is unintentional. This
 41798 volume of POSIX.1-2008 defers to the ISO C standard.

41799 If *s* is not a null pointer, *mbrlen()* shall determine the number of bytes constituting the character
 41800 pointed to by *s*. It shall be equivalent to:

41801 `mbstate_t internal;`
 41802 `mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);`

41803 If *ps* is a null pointer, the *mbrlen()* function shall use its own internal **mbstate_t** object, which is
 41804 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object
 41805 pointed to by *ps* shall be used to completely describe the current conversion state of the
 41806 associated character sequence. The implementation shall behave as if no function defined in this
 41807 volume of POSIX.1-2008 calls *mbrlen()*.

41808 The behavior of this function is affected by the *LC_CTYPE* category of the current locale.

41809 **RETURN VALUE**

41810 The *mbrlen()* function shall return the first of the following that applies:

41811	0	If the next <i>n</i> or fewer bytes complete the character that corresponds to the null wide character.
41812		
41813	<i>positive</i>	If the next <i>n</i> or fewer bytes complete a valid character; the value returned shall be the number of bytes that complete the character.
41814		
41815	(size_t)-2	If the next <i>n</i> bytes contribute to an incomplete but potentially valid character, and all <i>n</i> bytes have been processed. When <i>n</i> has at least the value of the {MB_CUR_MAX} macro, this case can only occur if <i>s</i> points at a sequence of redundant shift sequences (for implementations with state-dependent encodings).
41816		
41817		
41818		
41819		
41820	(size_t)-1	If an encoding error occurs, in which case the next <i>n</i> or fewer bytes do not contribute to a complete and valid character. In this case, [EILSEQ] shall be stored in <i>errno</i> and the conversion state is undefined.
41821		
41822		

41823 **ERRORS**

41824 The *mbrlen()* function shall fail if:

41825 [EILSEQ] An invalid character sequence is detected.

41826 The *mbrlen()* function may fail if:

41827 [EINVAL] *ps* points to an object that contains an invalid conversion state.

41828 **EXAMPLES**

41829 None.

41830 **APPLICATION USAGE**

41831 None.

41832 **RATIONALE**

41833 None.

41834 **FUTURE DIRECTIONS**

41835 None.

41836 **SEE ALSO**41837 [*mbsinit\(\)*](#), [*mbrtowc\(\)*](#)41838 XBD [*<wchar.h>*](#)41839 **CHANGE HISTORY**41840 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
41841 (E).41842 **Issue 6**41843 The *mbrlen()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.41844 **Issue 7**

41845 Austin Group Interpretation 1003.1-2001 #170 is applied.

mbrtowc()*System Interfaces*41846 **NAME**41847 **mbrtowc** — convert a character to a wide-character code (restartable)41848 **SYNOPSIS**41849 `#include <wchar.h>`

```
41850     size_t mbrtowc(wchar_t *restrict pwc, const char *restrict s,
41851                   size_t n, mbstate_t *restrict ps);
```

41852 **DESCRIPTION**

41853 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 41854 conflict between the requirements described here and the ISO C standard is unintentional. This
 41855 volume of POSIX.1-2008 defers to the ISO C standard.

41856 If *s* is a null pointer, the *mbrtowc()* function shall be equivalent to the call:

```
41857 mbrtowc(NULL, "", 1, ps)
```

41858 In this case, the values of the arguments *pwc* and *n* are ignored.

41859 If *s* is not a null pointer, the *mbrtowc()* function shall inspect at most *n* bytes beginning at the
 41860 byte pointed to by *s* to determine the number of bytes needed to complete the next character
 41861 (including any shift sequences). If the function determines that the next character is completed,
 41862 it shall determine the value of the corresponding wide character and then, if *pwc* is not a null
 41863 pointer, shall store that value in the object pointed to by *pwc*. If the corresponding wide
 41864 character is the null wide character, the resulting state described shall be the initial conversion
 41865 state.

41866 If *ps* is a null pointer, the *mbrtowc()* function shall use its own internal **mbstate_t** object, which
 41867 shall be initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t**
 41868 object pointed to by *ps* shall be used to completely describe the current conversion state of the
 41869 associated character sequence. The implementation shall behave as if no function defined in this
 41870 volume of POSIX.1-2008 calls *mbrtowc()*.

41871 The behavior of this function is affected by the *LC_CTYPE* category of the current locale.

41872 **RETURN VALUE**

41873 The *mbrtowc()* function shall return the first of the following that applies:

41874 **0** If the next *n* or fewer bytes complete the character that corresponds to the null
 41875 wide character (which is the value stored).

41876 between 1 and *n* inclusive

41877 If the next *n* or fewer bytes complete a valid character (which is the value
 41878 stored); the value returned shall be the number of bytes that complete the
 41879 character.

41880 **(size_t)–2** If the next *n* bytes contribute to an incomplete but potentially valid character,
 41881 and all *n* bytes have been processed (no value is stored). When *n* has at least
 41882 the value of the {MB_CUR_MAX} macro, this case can only occur if *s* points at
 41883 a sequence of redundant shift sequences (for implementations with state-
 41884 dependent encodings).

41885 **(size_t)–1** If an encoding error occurs, in which case the next *n* or fewer bytes do not
 41886 contribute to a complete and valid character (no value is stored). In this case,
 41887 [EILSEQ] shall be stored in *errno* and the conversion state is undefined.

41888 **ERRORS**41889 The *mbrtowc()* function shall fail if:

41890 [EILSEQ] An invalid character sequence is detected.

41891 The *mbrtowc()* function may fail if:41892 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.41893 **EXAMPLES**

41894 None.

41895 **APPLICATION USAGE**

41896 None.

41897 **RATIONALE**

41898 None.

41899 **FUTURE DIRECTIONS**

41900 None.

41901 **SEE ALSO**41902 *mbsinit()*, *mbsrtowcs()*

41903 XBD <wchar.h>

41904 **CHANGE HISTORY**41905 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
41906 (E).41907 **Issue 6**41908 The *mbrtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.41909 The following new requirements on POSIX implementations derive from alignment with the
41910 Single UNIX Specification:

- 41911
- The [EINVAL] error condition is added.

41912 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

41913 **Issue 7**

41914 Austin Group Interpretation 1003.1-2001 #170 is applied.

mbsinit()*System Interfaces*41915 **NAME**41916 **mbsinit** — determine conversion object status41917 **SYNOPSIS**41918 `#include <wchar.h>`41919 `int mbsinit(const mbstate_t *ps);`41920 **DESCRIPTION**

41921 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 41922 conflict between the requirements described here and the ISO C standard is unintentional. This
 41923 volume of POSIX.1-2008 defers to the ISO C standard.

41924 If *ps* is not a null pointer, the *mbsinit()* function shall determine whether the object pointed to by
 41925 *ps* describes an initial conversion state.

41926 **RETURN VALUE**

41927 The *mbsinit()* function shall return non-zero if *ps* is a null pointer, or if the pointed-to object
 41928 describes an initial conversion state; otherwise, it shall return zero.

41929 If an **mbstate_t** object is altered by any of the functions described as “restartable”, and is then
 41930 used with a different character sequence, or in the other conversion direction, or with a different
 41931 *LC_CTYPE* category setting than on earlier function calls, the behavior is undefined.

41932 **ERRORS**

41933 No errors are defined.

41934 **EXAMPLES**

41935 None.

41936 **APPLICATION USAGE**

41937 The **mbstate_t** object is used to describe the current conversion state from a particular character
 41938 sequence to a wide-character sequence (or *vice versa*) under the rules of a particular setting of the
 41939 *LC_CTYPE* category of the current locale.

41940 The initial conversion state corresponds, for a conversion in either direction, to the beginning of
 41941 a new character sequence in the initial shift state. A zero valued **mbstate_t** object is at least one
 41942 way to describe an initial conversion state. A zero valued **mbstate_t** object can be used to initiate
 41943 conversion involving any character sequence, in any *LC_CTYPE* category setting.

41944 **RATIONALE**

41945 None.

41946 **FUTURE DIRECTIONS**

41947 None.

41948 **SEE ALSO**41949 *mbrlen()*, *mbrtowc()*, *mbsrtowcs()*, *wcrtomb()*, *wcsrtombs()*41950 XBD *<wchar.h>*41951 **CHANGE HISTORY**

41952 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 41953 (E).

NAME

mbsnrtowcs, mbsrtowcs — convert a character string to a wide-character string (restartable)

SYNOPSIS

```
#include <wchar.h>
```

```
size_t mbsnrtowcs(wchar_t *restrict dst, const char **restrict src,
                  size_t nmc, size_t len, mbstate_t *restrict ps);
size_t mbsrtowcs(wchar_t *restrict dst, const char **restrict src,
                  size_t len, mbstate_t *restrict ps);
```

DESCRIPTION

For **mbsrtowcs()**: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The **mbsrtowcs()** function shall convert a sequence of characters, beginning in the conversion state described by the object pointed to by *ps*, from the array indirectly pointed to by *src* into a sequence of corresponding wide characters. If *dst* is not a null pointer, the converted characters shall be stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null character, which shall also be stored. Conversion shall stop early in either of the following cases:

- A sequence of bytes is encountered that does not form a valid character.
- *len* codes have been stored into the array pointed to by *dst* (and *dst* is not a null pointer).

Each conversion shall take place as if by a call to the **mbrtowc()** function.

If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last character converted (if any). If conversion stopped due to reaching a terminating null character, and if *dst* is not a null pointer, the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the **mbsrtowcs()** function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence.

The **mbsnrtowcs()** function shall be equivalent to the **mbsrtowcs()** function, except that the conversion of characters pointed to by *src* is limited to at most *nmc* bytes (the size of the input buffer).

The behavior of these functions shall be affected by the **LC_CTYPE** category of the current locale.

The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls these functions.

RETURN VALUE

If the input conversion encounters a sequence of bytes that do not form a valid character, an encoding error occurs. In this case, these functions shall store the value of the macro **[EILSEQ]** in **errno** and shall return **(size_t)-1**; the conversion state is undefined. Otherwise, these functions shall return the number of characters successfully converted, not including the terminating null (if any).

mbsrtowcs()*System Interfaces*41996 **ERRORS**

41997 These functions shall fail if:

41998 [EILSEQ] An invalid character sequence is detected.

41999 These functions may fail if:

42000 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.42001 **EXAMPLES**

42002 None.

42003 **APPLICATION USAGE**

42004 None.

42005 **RATIONALE**

42006 None.

42007 **FUTURE DIRECTIONS**

42008 None.

42009 **SEE ALSO**42010 *iconv()*, *mbrtowc()*, *mbsinit()*42011 XBD <*wchar.h*>42012 **CHANGE HISTORY**42013 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
42014 (E).42015 **Issue 6**42016 The *mbsrtowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

42017 The [EINVAL] error condition is marked CX.

42018 **Issue 7**

42019 Austin Group Interpretation 1003.1-2001 #170 is applied.

42020 The *mbsnrtowcs()* function is added from The Open Group Technical Standard, 2006, Extended
42021 API Set Part 1.

System Interfaces

mbstowcs()**NAME**

mbstowcs — convert a character string to a wide-character string

SYNOPSIS

```
#include <stdlib.h>
```

```
size_t mbstowcs(wchar_t *restrict pwcs, const char *restrict s,  
size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *mbstowcs()* function shall convert a sequence of characters that begins in the initial shift state from the array pointed to by *s* into a sequence of corresponding wide-character codes and shall store not more than *n* wide-character codes into the array pointed to by *pwcs*. No characters that follow a null byte (which is converted into a wide-character code with value 0) shall be examined or converted. Each character shall be converted as if by a call to *mbtowc()*, except that the shift state of *mbtowc()* is not affected.

No more than *n* elements shall be modified in the array pointed to by *pwcs*. If copying takes place between objects that overlap, the behavior is undefined.

XSI The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale. If *pwcs* is a null pointer, *mbstowcs()* shall return the length required to convert the entire array regardless of the value of *n*, but no values are stored.

RETURN VALUE

CX If an invalid character is encountered, *mbstowcs()* shall return **(size_t)-1** and may set *errno* to indicate the error.

XSI Otherwise, *mbstowcs()* shall return the number of the array elements modified (or required if *pwcs* is null), not including a terminating 0 code, if any. The array shall not be zero-terminated if the value returned is *n*.

ERRORS

The *mbstowcs()* function shall fail if:

XSI **[EILSEQ]** An invalid byte sequence is detected.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mblen(), *mbtowc()*, *wctomb()*, *wcstombs()*

XBD *<stdlib.h>*

mbstowcs()*System Interfaces*42063 **CHANGE HISTORY**

42064 First released in Issue 4. Aligned with the ISO C standard.

42065 **Issue 6**42066 The *mbstowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

42067 Extensions beyond the ISO C standard are marked.

42068 **Issue 7**

42069 Austin Group Interpretation 1003.1-2001 #170 is applied.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

mbtowc()**NAME**

mbtowc — convert a character to a wide-character code

SYNOPSIS

```
#include <stdlib.h>
```

```
int mbtowc(wchar_t *restrict pwc, const char *restrict s, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

If *s* is not a null pointer, *mbtowc()* shall determine the number of bytes that constitute the character pointed to by *s*. It shall then determine the wide-character code for the value of type **wchar_t** that corresponds to that character. (The value of the wide-character code corresponding to the null byte is 0.) If the character is valid and *pwc* is not a null pointer, *mbtowc()* shall store the wide-character code in the object pointed to by *pwc*.

The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a state-dependent encoding, this function is placed into its initial state by a call for which its character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a null pointer shall cause this function to return a non-zero value if encodings have state dependency, and 0 otherwise. If the implementation employs special bytes to change the shift state, these bytes shall not produce separate wide-character codes, but shall be grouped with an adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be unspecified. At most *n* bytes of the array pointed to by *s* shall be examined.

The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls *mbtowc()*.

RETURN VALUE

If *s* is a null pointer, *mbtowc()* shall return a non-zero or 0 value, if character encodings, respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc()* shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the converted character (if the next *n* or fewer bytes form a valid character), or return -1 and may set *errno* to indicate the error (if they do not form a valid character).

In no case shall the value returned be greater than *n* or the value of the {MB_CUR_MAX} macro.

ERRORS

The *mbtowc()* function shall fail if:

XSI	[EILSEQ]	An invalid character sequence is detected.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

mbtowc()*System Interfaces*42113 **SEE ALSO**42114 [*mblen\(\)*](#), [*mbstowcs\(\)*](#), [*wctomb\(\)*](#), [*wcstombs\(\)*](#)42115 XBD [*<stdlib.h>*](#)42116 **CHANGE HISTORY**

42117 First released in Issue 4. Aligned with the ISO C standard.

42118 **Issue 6**42119 The *mbtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

42120 Extensions beyond the ISO C standard are marked.

42121 **Issue 7**

42122 Austin Group Interpretation 1003.1-2001 #170 is applied.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

42123 NAME

42124 memcpy — copy bytes in memory

42125 SYNOPSIS

```
42126 XSI      #include <string.h>
42127
42127 void *memcpy(void *restrict s1, const void *restrict s2,
42128             int c, size_t n);
```

42129 DESCRIPTION

42130 The *memcpy()* function shall copy bytes from memory area *s2* into *s1*, stopping after the first
 42131 occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied,
 42132 whichever comes first. If copying takes place between objects that overlap, the behavior is
 42133 undefined.

42134 RETURN VALUE

42135 The *memcpy()* function shall return a pointer to the byte after the copy of *c* in *s1*, or a null
 42136 pointer if *c* was not found in the first *n* bytes of *s2*.

42137 ERRORS

42138 No errors are defined.

42139 EXAMPLES

42140 None.

42141 APPLICATION USAGE

42142 The *memcpy()* function does not check for the overflow of the receiving memory area.

42143 RATIONALE

42144 None.

42145 FUTURE DIRECTIONS

42146 None.

42147 SEE ALSO

42148 XBD [<string.h>](#)

42149 CHANGE HISTORY

42150 First released in Issue 1. Derived from Issue 1 of the SVID.

42151 Issue 6

42152 The **restrict** keyword is added to the *memcpy()* prototype for alignment with the
 42153 ISO/IEC 9899:1999 standard.

memchr()*System Interfaces*42154 **NAME**42155 **memchr** — find byte in memory42156 **SYNOPSIS**42157 `#include <string.h>`42158 `void *memchr(const void *s, int c, size_t n);`42159 **DESCRIPTION**

42160 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42161 conflict between the requirements described here and the ISO C standard is unintentional. This
 42162 volume of POSIX.1-2008 defers to the ISO C standard.

42163 The *memchr()* function shall locate the first occurrence of *c* (converted to an **unsigned char**) in
 42164 the initial *n* bytes (each interpreted as **unsigned char**) of the object pointed to by *s*.

42165 **RETURN VALUE**

42166 The *memchr()* function shall return a pointer to the located byte, or a null pointer if the byte does
 42167 not occur in the object.

42168 **ERRORS**

42169 No errors are defined.

42170 **EXAMPLES**

42171 None.

42172 **APPLICATION USAGE**

42173 None.

42174 **RATIONALE**

42175 None.

42176 **FUTURE DIRECTIONS**

42177 None.

42178 **SEE ALSO**42179 XBD [<string.h>](#)42180 **CHANGE HISTORY**

42181 First released in Issue 1. Derived from Issue 1 of the SVID.

42182 **NAME**

42183 memcmp — compare bytes in memory

42184 **SYNOPSIS**

42185 #include <string.h>

42186 int memcmp(const void *s1, const void *s2, size_t n);

42187 **DESCRIPTION**

42188 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 42189 conflict between the requirements described here and the ISO C standard is unintentional. This
 42190 volume of POSIX.1-2008 defers to the ISO C standard.

42191 The *memcmp()* function shall compare the first *n* bytes (each interpreted as **unsigned char**) of the
 42192 object pointed to by *s1* to the first *n* bytes of the object pointed to by *s2*.

42193 The sign of a non-zero return value shall be determined by the sign of the difference between the
 42194 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the objects
 42195 being compared.

42196 **RETURN VALUE**

42197 The *memcmp()* function shall return an integer greater than, equal to, or less than 0, if the object
 42198 pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*, respectively.

42199 **ERRORS**

42200 No errors are defined.

42201 **EXAMPLES**

42202 None.

42203 **APPLICATION USAGE**

42204 None.

42205 **RATIONALE**

42206 None.

42207 **FUTURE DIRECTIONS**

42208 None.

42209 **SEE ALSO**

42210 XBD <string.h>

42211 **CHANGE HISTORY**

42212 First released in Issue 1. Derived from Issue 1 of the SVID.

memcpy()

System Interfaces

42213 **NAME**42214 `memcpy` — copy bytes in memory42215 **SYNOPSIS**42216 `#include <string.h>`42217 `void *memcpy(void *restrict s1, const void *restrict s2, size_t n);`42218 **DESCRIPTION**

42219 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42220 conflict between the requirements described here and the ISO C standard is unintentional. This
 42221 volume of POSIX.1-2008 defers to the ISO C standard.

42222 The *memcpy()* function shall copy *n* bytes from the object pointed to by *s2* into the object pointed
 42223 to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

42224 **RETURN VALUE**42225 The *memcpy()* function shall return *s1*; no return value is reserved to indicate an error.42226 **ERRORS**

42227 No errors are defined.

42228 **EXAMPLES**

42229 None.

42230 **APPLICATION USAGE**42231 The *memcpy()* function does not check for the overflow of the receiving memory area.42232 **RATIONALE**

42233 None.

42234 **FUTURE DIRECTIONS**

42235 None.

42236 **SEE ALSO**42237 XBD [<string.h>](#)42238 **CHANGE HISTORY**

42239 First released in Issue 1. Derived from Issue 1 of the SVID.

42240 **Issue 6**42241 The *memcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

42242 NAME

42243 memmove — copy bytes in memory with overlapping areas

42244 SYNOPSIS

42245 #include <string.h>

42246 void *memmove(void *s1, const void *s2, size_t n);

42247 DESCRIPTION

42248 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42249 conflict between the requirements described here and the ISO C standard is unintentional. This
 42250 volume of POSIX.1-2008 defers to the ISO C standard.

42251 The *memmove()* function shall copy *n* bytes from the object pointed to by *s2* into the object
 42252 pointed to by *s1*. Copying takes place as if the *n* bytes from the object pointed to by *s2* are first
 42253 copied into a temporary array of *n* bytes that does not overlap the objects pointed to by *s1* and
 42254 *s2*, and then the *n* bytes from the temporary array are copied into the object pointed to by *s1*.

42255 RETURN VALUE

42256 The *memmove()* function shall return *s1*; no return value is reserved to indicate an error.

42257 ERRORS

42258 No errors are defined.

42259 EXAMPLES

42260 None.

42261 APPLICATION USAGE

42262 None.

42263 RATIONALE

42264 None.

42265 FUTURE DIRECTIONS

42266 None.

42267 SEE ALSO

42268 XBD <string.h>

42269 CHANGE HISTORY

42270 First released in Issue 4. Derived from the ANSI C standard.

memset()

System Interfaces

42271 **NAME**42272 **memset** — set bytes in memory42273 **SYNOPSIS**

42274 #include <string.h>

42275 void *memset(void *s, int c, size_t n);

42276 **DESCRIPTION**

42277 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42278 conflict between the requirements described here and the ISO C standard is unintentional. This
 42279 volume of POSIX.1-2008 defers to the ISO C standard.

42280 The *memset()* function shall copy *c* (converted to an **unsigned char**) into each of the first *n* bytes
 42281 of the object pointed to by *s*.

42282 **RETURN VALUE**42283 The *memset()* function shall return *s*; no return value is reserved to indicate an error.42284 **ERRORS**

42285 No errors are defined.

42286 **EXAMPLES**

42287 None.

42288 **APPLICATION USAGE**

42289 None.

42290 **RATIONALE**

42291 None.

42292 **FUTURE DIRECTIONS**

42293 None.

42294 **SEE ALSO**

42295 XBD <string.h>

42296 **CHANGE HISTORY**

42297 First released in Issue 1. Derived from Issue 1 of the SVID.

NAME

mkdir, mkdirat — make a directory relative to directory file descriptor

SYNOPSIS

```
#include <sys/stat.h>

int mkdir(const char *path, mode_t mode);
int mkdirat(int fd, const char *path, mode_t mode);
```

DESCRIPTION

The *mkdir()* function shall create a new directory with name *path*. The file permission bits of the new directory shall be initialized from *mode*. These file permission bits of the *mode* argument shall be modified by the process' file creation mask.

When bits in *mode* other than the file permission bits are set, the meaning of these additional bits is implementation-defined.

The directory's user ID shall be set to the process' effective user ID. The directory's group ID shall be set to the group ID of the parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the directory's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the directory's group ID to the effective group ID of the calling process.

The newly created directory shall be an empty directory.

If *path* names a symbolic link, *mkdir()* shall fail and set *errno* to [EEXIST].

Upon successful completion, *mkdir()* shall mark for update the last data access, last data modification, and last file status change timestamps of the directory. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

The *mkdirat()* function shall be equivalent to the *mkdir()* function except in the case where *path* specifies a relative path. In this case the newly created directory is created relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function shall not perform the check.

If *mkdirat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working directory is used and the behavior shall be identical to a call to *mkdir()*.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return -1 and set *errno* to indicate the error. If -1 is returned, no directory shall be created.

ERRORS

These functions shall fail if:

[EACCES]	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created.
[EEXIST]	The named file exists.
[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
[EMLINK]	The link count of the parent directory would exceed {LINK_MAX}.

mkdir()*System Interfaces*

42340	[ENAMETOOLONG]	
42341		The length of a component of a pathname is longer than {NAME_MAX}.
42342	[ENOENT]	A component of the path prefix specified by <i>path</i> does not name an existing directory or <i>path</i> is an empty string.
42343		
42344	[ENOSPC]	The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.
42345		
42346	[ENOTDIR]	A component of the path prefix is not a directory.
42347	[EROFS]	The parent directory resides on a read-only file system.
42348		In addition, the <i>mkdirat()</i> function shall fail if:
42349	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading.
42350		
42351		These functions may fail if:
42352	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
42353		
42354	[ENAMETOOLONG]	
42355		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
42356		
42357		
42358		The <i>mkdirat()</i> function may fail if:
42359	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is neither AT_FDCWD nor a file descriptor associated with a directory.
42360		

EXAMPLES**Creating a Directory**

The following example shows how to create a directory named **/home/cnd/mod1**, with read/write/search permissions for owner and group, and with read/search permissions for others.

```
#include <sys/types.h>
#include <sys/stat.h>

int status;
...
status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
```

APPLICATION USAGE

None.

RATIONALE

The *mkdir()* function originated in 4.2 BSD and was added to System V in Release 3.0.

4.3 BSD detects [ENAMETOOLONG].

The POSIX.1-1990 standard required that the group ID of a newly created directory be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not

assume which group ID will be used. If it matters, an application can use *chown()* to set the group ID after the directory is created, or determine under what conditions the implementation will set the desired group ID.

The purpose of the *mkdirat()* function is to create a directory in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to the call to *mkdir()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *mkdirat()* function it can be guaranteed that the newly created directory is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod(), *mkdtemp()*, *mknod()*, *umask()*

XBD [<sys/stat.h>](#), [<sys/types.h>](#)

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The *mkdirat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

mkdtemp()*System Interfaces*42412 **NAME**

42413 mkdtemp, mkstemp — create a unique directory or file

42414 **SYNOPSIS**

```

42415 CX      #include <stdlib.h>
42416          char *mkdtemp(char *template);
42417          int mkstemp(char *template);

```

42418 **DESCRIPTION**

42419 The *mkdtemp()* function uses the contents of *template* to construct a unique directory name. The
 42420 string provided in *template* shall be a filename ending with six trailing 'X's. The *mkdtemp()*
 42421 function shall replace each 'X' with a character from the portable filename character set. The
 42422 characters are chosen such that the resulting name does not duplicate the name of an existing file
 42423 at the time of a call to *mkdtemp()*. The unique directory name is used to attempt to create the
 42424 directory using mode 0700 as modified by the file creation mask.

42425 The *mkstemp()* function shall replace the contents of the string pointed to by *template* by a unique
 42426 filename, and return a file descriptor for the file open for reading and writing. The *mkstemp()*
 42427 function shall create the file, and obtain a file descriptor for it, as if by a call to:

```

42428 open(filename, O_RDWR|O_CREAT|O_EXCL, S_IRUSR|S_IWUSR)

```

42429 The function thus prevents any possible race condition between testing whether the file exists
 42430 and opening it for use. The string in *template* should look like a filename with six trailing 'X's;
 42431 *mkstemp()* replaces each 'X' with a character from the portable filename character set. The
 42432 characters are chosen such that the resulting name does not duplicate the name of an existing file
 42433 at the time of a call to *mkstemp()*.

42434 **RETURN VALUE**

42435 Upon successful completion, the *mkdtemp()* function shall return a pointer to the string
 42436 containing the directory name if it was created. Otherwise, it shall return a null pointer and shall
 42437 set *errno* to indicate the error.

42438 Upon successful completion, the *mkstemp()* function shall return an open file descriptor.
 42439 Otherwise, it shall return -1 if no suitable file could be created.

42440 **ERRORS**

42441 The *mkdtemp()* function shall fail if:

42442 42443	[EACCES]	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created.
42444	[EINVAL]	The string pointed to by <i>template</i> does not end in "XXXXXX".
42445 42446	[ELOOP]	A loop exists in symbolic links encountered during resolution of the path of the directory to be created.
42447	[EMLINK]	The link count of the parent directory would exceed {LINK_MAX}.
42448 42449	[ENAMETOOLONG]	The length of a component of a pathname is longer than {NAME_MAX}.
42450 42451	[ENOENT]	A component of the path prefix specified by the <i>template</i> argument does not name an existing directory.
42452 42453	[ENOSPC]	The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.

- 42454 [ENOTDIR] A component of the path prefix is not a directory.
- 42455 [EROFS] The parent directory resides on a read-only file system.
- 42456 The *mkdtemp()* function may fail if:
- 42457 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
- 42458 resolution of the path of the directory to be created.
- 42459 [ENAMETOOLONG]
- 42460 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
- 42461 symbolic link produced an intermediate result with a length that exceeds
- 42462 {PATH_MAX}.
- 42463 The error conditions for the *mkstemp()* function are defined in *open()*.

42464 EXAMPLES

42465 Generating a Filename

42466 The following example creates a file with a 10-character name beginning with the characters

42467 "file" and opens the file for reading and writing. The value returned as the value of *fd* is a file

42468 descriptor that identifies the file.

```
42469 #include <stdlib.h>
42470 ...
42471 char template[] = "/tmp/fileXXXXXX";
42472 int fd;
42473
42474 fd = mkstemp(template);
```

42474 APPLICATION USAGE

42475 It is possible to run out of letters.

42476 The *mkdtemp()* and *mkstemp()* functions need not check to determine whether the filename part

42477 of *template* exceeds the maximum allowable filename length.

42478 RATIONALE

42479 None.

42480 FUTURE DIRECTIONS

42481 None.

42482 SEE ALSO

42483 *getpid()*, *mkdir()*, *open()*, *tmpfile()*, *tmpnam()*

42484 XBD <stdlib.h>

42485 CHANGE HISTORY

42486 First released in Issue 4, Version 2.

42487 Issue 5

42488 Moved from X/OPEN UNIX extension to BASE.

mkdtemp()*System Interfaces*42489 **Issue 7**

42490 Austin Group Interpretation 1003.1-2001 #143 is applied.

42491 SD5-XSH-ERN-168 is applied, clarifying file permissions upon creation.

42492 The *mkstemp()* function is moved from the XSI option to the Base.42493 The *mkdtemp()* function is added from The Open Group Technical Standard, 2006, Extended API
42494 Set Part 1.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

mkfifo, mkfifoat — make a FIFO special file relative to directory file descriptor

SYNOPSIS

```
#include <sys/stat.h>
```

```
int mkfifo(const char *path, mode_t mode);
```

```
int mkfifoat(int fd, const char *path, mode_t mode);
```

DESCRIPTION

The *mkfifo()* function shall create a new FIFO special file named by the pathname pointed to by *path*. The file permission bits of the new FIFO shall be initialized from *mode*. The file permission bits of the *mode* argument shall be modified by the process' file creation mask.

When bits in *mode* other than the file permission bits are set, the effect is implementation-defined.

If *path* names a symbolic link, *mkfifo()* shall fail and set *errno* to [EEXIST].

The FIFO's user ID shall be set to the process' effective user ID. The FIFO's group ID shall be set to the group ID of the parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the FIFO's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the FIFO's group ID to the effective group ID of the calling process.

Upon successful completion, *mkfifo()* shall mark for update the last data access, last data modification, and last file status change timestamps of the file. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

The *mkfifoat()* function shall be equivalent to the *mkfifo()* function except in the case where *path* specifies a relative path. In this case the newly created FIFO is created relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function shall not perform the check.

If *mkfifoat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working directory is used and the behavior shall be identical to a call to *mkfifo()*.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return -1 and set *errno* to indicate the error. If -1 is returned, no FIFO shall be created.

ERRORS

These functions shall fail if:

[EACCES] A component of the path prefix denies search permission, or write permission is denied on the parent directory of the FIFO to be created.

[EEXIST] The named file already exists.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]

The length of a component of a pathname is longer than {NAME_MAX}.

mkfifo()*System Interfaces*

42537	[ENOENT]	A component of the path prefix specified by <i>path</i> does not name an existing directory or <i>path</i> is an empty string.
42538		
42539	[ENOSPC]	The directory that would contain the new file cannot be extended or the file system is out of file-allocation resources.
42540		
42541	[ENOTDIR]	A component of the path prefix is not a directory.
42542	[EROFS]	The named file resides on a read-only file system.
42543		The <i>mkfifoat()</i> function shall fail if:
42544	[EACCES]	<i>fd</i> was not opened with O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
42545		
42546	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
42547		
42548		These functions may fail if:
42549	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
42550		
42551	[ENAMETOOLONG]	
42552		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
42553		
42554		
42555		The <i>mkfifoat()</i> function may fail if:
42556	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is neither AT_FDCWD nor a file descriptor associated with a directory.
42557		

EXAMPLES**Creating a FIFO File**

The following example shows how to create a FIFO file named **/home/cnd/mod_done**, with read/write permissions for owner, and with read permissions for group and others.

```
#include <sys/types.h>
#include <sys/stat.h>

int status;
...
status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
S_IRGRP | S_IROTH);
```

APPLICATION USAGE

None.

RATIONALE

The syntax of this function is intended to maintain compatibility with historical implementations of *mknod()*. The latter function was included in the 1984 /usr/group standard but only for use in creating FIFO special files. The *mknod()* function was originally excluded from the POSIX.1-1988 standard as implementation-defined and replaced by *mkdir()* and *mkfifo()*. The *mknod()* function is now included for alignment with the Single UNIX Specification.

The POSIX.1-1990 standard required that the group ID of a newly created FIFO be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2

required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown()* to set the group ID after the FIFO is created, or determine under what conditions the implementation will set the desired group ID.

The purpose of the *mkfifoat()* function is to create a FIFO special file in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *mkfifo()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *mkfifoat()* function it can be guaranteed that the newly created FIFO is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod(), *mknod()*, *umask()*

XBD *<sys/stat.h>*, *<sys/types.h>*

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the SYNOPSIS, the optional include of the *<sys/types.h>* header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The *mkfifoat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

mknod()*System Interfaces***NAME**

mknod, mknodat — make directory, special file, or regular file

SYNOPSIS

```

#include <sys/stat.h>

int mknod(const char *path, mode_t mode, dev_t dev);
int mknodat(int fd, const char *path, mode_t mode, dev_t dev);

```

DESCRIPTION

The *mknod()* function shall create a new file named by the pathname to which the argument *path* points.

The file type for *path* is OR'ed into the *mode* argument, and the application shall select one of the following symbolic constants:

Name	Description
S_IFIFO	FIFO-special
S_IFCHR	Character-special (non-portable)
S_IFDIR	Directory (non-portable)
S_IFBLK	Block-special (non-portable)
S_IFREG	Regular (non-portable)

The only portable use of *mknod()* is to create a FIFO-special file. If *mode* is not S_IFIFO or *dev* is not 0, the behavior of *mknod()* is unspecified.

The permissions for the new file are OR'ed into the *mode* argument, and may be selected from any combination of the following symbolic constants:

Name	Description
S_ISUID	Set user ID on execution.
S_ISGID	Set group ID on execution.
S_IRWXU	Read, write, or execute (search) by owner.
S_IRUSR	Read by owner.
S_IWUSR	Write by owner.
S_IXUSR	Execute (search) by owner.
S_IRWXG	Read, write, or execute (search) by group.
S_IRGRP	Read by group.
S_IWGRP	Write by group.
S_IXGRP	Execute (search) by group.
S_IRWXO	Read, write, or execute (search) by others.
S_IROTH	Read by others.
S_IWOTH	Write by others.
S_IXOTH	Execute (search) by others.
S_ISVTX	On directories, restricted deletion flag.

The user ID of the file shall be initialized to the effective user ID of the process. The group ID of the file shall be initialized to either the effective group ID of the process or the group ID of the parent directory. Implementations shall provide a way to initialize the file's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the file's group ID to the effective group ID of the calling process. The owner, group, and other permission bits of *mode* shall be modified by the file mode creation mask of the process. The *mknod()* function shall clear each bit whose corresponding bit in the file mode creation mask of the process is set.

- 42659 If *path* names a symbolic link, *mknod()* shall fail and set *errno* to [EEXIST].
- 42660 Upon successful completion, *mknod()* shall mark for update the last data access, last data
42661 modification, and last file status change timestamps of the file. Also, the last data modification
42662 and last file status change timestamps of the directory that contains the new entry shall be
42663 marked for update.
- 42664 Only a process with appropriate privileges may invoke *mknod()* for file types other than FIFO-
42665 special.
- 42666 The *mknodat()* function shall be equivalent to the *mknod()* function except in the case where *path*
42667 specifies a relative path. In this case the newly created directory, special file, or regular file is
42668 located relative to the directory associated with the file descriptor *fd* instead of the current
42669 working directory. If the file descriptor was opened without O_SEARCH, the function shall
42670 check whether directory searches are permitted using the current permissions of the directory
42671 underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function
42672 shall not perform the check.
- 42673 If *mknodat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
42674 directory is used and the behavior shall be identical to a call to *mknod()*.
- 42675 **RETURN VALUE**
- 42676 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
42677 return -1 and set *errno* to indicate the error. If -1 is returned, the new file shall not be created.
- 42678 **ERRORS**
- 42679 These functions shall fail if:
- | | |
|----------------------|---|
| 42680 [EACCES] | A component of the path prefix denies search permission, or write permission
42681 is denied on the parent directory. |
| 42682 [EEXIST] | The named file exists. |
| 42683 [EINVAL] | An invalid argument exists. |
| 42684 [EIO] | An I/O error occurred while accessing the file system. |
| 42685 [ELOOP] | A loop exists in symbolic links encountered during resolution of the <i>path</i>
42686 argument. |
| 42687 [ENAMETOOLONG] | The length of a component of a pathname is longer than {NAME_MAX}. |
| 42689 [ENOENT] | A component of the path prefix specified by <i>path</i> does not name an existing
42690 directory or <i>path</i> is an empty string. |
| 42691 [ENOSPC] | The directory that would contain the new file cannot be extended or the file
42692 system is out of file allocation resources. |
| 42693 [ENOTDIR] | A component of the path prefix is not a directory. |
| 42694 [EPERM] | The invoking process does not have appropriate privileges and the file type is
42695 not FIFO-special. |
| 42696 [EROFS] | The directory in which the file is to be created is located on a read-only file
42697 system. |

mknod()*System Interfaces*

- 42698 The *mknodat()* function shall fail if:
- 42699 [EACCES] *fd* was not opened with O_SEARCH and the permissions of the directory
42700 underlying *fd* do not permit directory searches.
- 42701 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
42702 neither AT_FDCWD nor a valid file descriptor open for reading or searching.
- 42703 These functions may fail if:
- 42704 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
42705 resolution of the *path* argument.
- 42706 [ENAMETOOLONG]
42707 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
42708 symbolic link produced an intermediate result with a length that exceeds
42709 {PATH_MAX}.
- 42710 The *mknodat()* function may fail if:
- 42711 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
42712 file descriptor associated with a directory.

EXAMPLES**Creating a FIFO Special File**

42715 The following example shows how to create a FIFO special file named */home/cnd/mod_done*,
42716 with read/write permissions for owner, and with read permissions for group and others.

```
42717 #include <sys/types.h>
42718 #include <sys/stat.h>
42719 dev_t dev;
42720 int status;
42721 ...
42722 status = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
42723 S_IRUSR | S_IRGRP | S_IROTH, dev);
```

APPLICATION USAGE

42725 The *mkfifo()* function is preferred over this function for making FIFO special files.

RATIONALE

42727 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group
42728 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required
42729 that implementations provide a way to have the group ID be set to the group ID of the
42730 containing directory, but did not prohibit implementations also supporting a way to set the
42731 group ID to the effective group ID of the creating process. Conforming applications should not
42732 assume which group ID will be used. If it matters, an application can use *chown()* to set the
42733 group ID after the file is created, or determine under what conditions the implementation will
42734 set the desired group ID.

42735 The purpose of the *mknodat()* function is to create directories, special files, or regular files in
42736 directories other than the current working directory without exposure to race conditions. Any
42737 part of the path of a file could be changed in parallel to a call to *mknod()*, resulting in unspecified
42738 behavior. By opening a file descriptor for the target directory and using the *mknodat()* function it
42739 can be guaranteed that the newly created directory, special file, or regular file is located relative
42740 to the desired directory.

42741 **FUTURE DIRECTIONS**

42742 None.

42743 **SEE ALSO**42744 *chmod()*, *creat()*, *exec*, *fstatat()*, *mkdir()*, *mkfifo()*, *open()*, *umask()*

42745 XBD <sys/stat.h>

42746 **CHANGE HISTORY**

42747 First released in Issue 4, Version 2.

42748 **Issue 5**

42749 Moved from X/OPEN UNIX extension to BASE.

42750 **Issue 6**

42751 The normative text is updated to avoid use of the term “must” for application requirements.

42752 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
42753 [ELOOP] error condition is added.42754 **Issue 7**

42755 Austin Group Interpretation 1003.1-2001 #143 is applied.

42756 The *mknodat()* function is added from The Open Group Technical Standard, 2006, Extended API
42757 Set Part 2.

42758 Changes are made related to support for finegrained timestamps.

42759 Changes are made to allow a directory to be opened for searching.

mkstemp()*System Interfaces*42760 **NAME**

42761 mkstemp — create a unique directory

42762 **SYNOPSIS**

```
42763 CX #include <stdlib.h>  
42764 int mkstemp(char *template);
```

42765 **DESCRIPTION**42766 Refer to *mkdtemp()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

mktime — convert broken-down time into time since the Epoch

SYNOPSIS

```
#include <time.h>

time_t mktime(struct tm *timeptr);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *mktime()* function shall convert the broken-down time, expressed as local time, in the structure pointed to by *timeptr*, into a time since the Epoch value with the same encoding as that of the values returned by *time()*. The original values of the *tm_wday* and *tm_yday* components of the structure are ignored, and the original values of the other components are not restricted to the ranges described in **<time.h>**.

A positive or 0 value for *tm_isdst* shall cause *mktime()* to presume initially that Daylight Savings Time, respectively, is or is not in effect for the specified time. A negative value for *tm_isdst* shall cause *mktime()* to attempt to determine whether Daylight Savings Time is in effect for the specified time.

Local timezone information shall be set as though *mktime()* called *tzset()*.

The relationship between the **tm** structure (defined in the **<time.h>** header) and the time in seconds since the Epoch is that the result shall be as specified in the expression given in the definition of seconds since the Epoch (see XBD Section 4.15, on page 113) corrected for timezone and any seasonal time adjustments, where the names in the structure and in the expression correspond.

Upon successful completion, the values of the *tm_wday* and *tm_yday* components of the structure shall be set appropriately, and the other components are set to represent the specified time since the Epoch, but with their values forced to the ranges indicated in the **<time.h>** entry; the final value of *tm_mday* shall not be set until *tm_mon* and *tm_year* are determined.

RETURN VALUE

The *mktime()* function shall return the specified time since the Epoch encoded as a value of type **time_t**. If the time since the Epoch cannot be represented, the function shall return the value **(time_t)-1** and may set *errno* to indicate the error.

ERRORS

The *mktime()* function may fail if:

[EOVERFLOW] The result cannot be represented.

EXAMPLES

What day of the week is July 4, 2001?

```
#include <stdio.h>
#include <time.h>

struct tm time_str;

char daybuf[20];

int main(void)
{
    time_str.tm_year = 2001 - 1900;
```

mktime()*System Interfaces*

```

42811         time_str.tm_mon = 7 - 1;
42812         time_str.tm_mday = 4;
42813         time_str.tm_hour = 0;
42814         time_str.tm_min = 0;
42815         time_str.tm_sec = 1;
42816         time_str.tm_isdst = -1;
42817         if (mktime(&time_str) == -1)
42818             (void)puts("-unknown-");
42819         else {
42820             (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);
42821             (void)puts(daybuf);
42822         }
42823         return 0;
42824     }

```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime(), clock(), ctime(), difftime(), gmtime(), localtime(), strftime(), strptime(), time(), tzset(), utime()

XBD Section 4.15 (on page 113), **<time.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ANSI C standard.

Issue 6

Extensions beyond the ISO C standard are marked.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/58 is applied, updating the RETURN VALUE and ERRORS sections to add the optional [Eoverflow] error as a CX extension.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/59 is applied, adding the *tzset()* function to the SEE ALSO section.

NAME

mlock, munlock — lock or unlock a range of process address space (**REALTIME**)

SYNOPSIS

```
#include <sys/mman.h>

int mlock(const void *addr, size_t len);
int munlock(const void *addr, size_t len);
```

DESCRIPTION

The *mlock()* function shall cause those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes to be memory-resident until unlocked or until the process exits or *execs* another process image. The implementation may require that *addr* be a multiple of {PAGESIZE}.

The *munlock()* function shall unlock those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes, regardless of how many times *mlock()* has been called by the process for any of the pages in the specified range. The implementation may require that *addr* be a multiple of {PAGESIZE}.

If any of the pages in the range specified to a call to *munlock()* are also mapped into the address spaces of other processes, any locks established on those pages by another process are unaffected by the call of this process to *munlock()*. If any of the pages in the range specified by a call to *munlock()* are also mapped into other portions of the address space of the calling process outside the range specified, any locks established on those pages via the other mappings are also unaffected by this call.

Upon successful return from *mlock()*, pages in the specified range shall be locked and memory-resident. Upon successful return from *munlock()*, pages in the specified range shall be unlocked with respect to the address space of the process. Memory residency of unlocked pages is unspecified.

Appropriate privileges are required to lock process memory with *mlock()*.

RETURN VALUE

Upon successful completion, the *mlock()* and *munlock()* functions shall return a value of zero. Otherwise, no change is made to any locks in the address space of the process, and the function shall return a value of -1 and set *errno* to indicate the error.

ERRORS

The *mlock()* and *munlock()* functions shall fail if:

[ENOMEM] Some or all of the address range specified by the *addr* and *len* arguments does not correspond to valid mapped pages in the address space of the process.

The *mlock()* function shall fail if:

[EAGAIN] Some or all of the memory identified by the operation could not be locked when the call was made.

The *mlock()* and *munlock()* functions may fail if:

[EINVAL] The *addr* argument is not a multiple of {PAGESIZE}.

The *mlock()* function may fail if:

[ENOMEM] Locking the pages mapped by the specified range would exceed an implementation-defined limit on the amount of memory that the process may lock.

mlock()*System Interfaces*

42887 [EPERM] The calling process does not have appropriate privileges to perform the
 42888 requested operation.

42889 **EXAMPLES**

42890 None.

42891 **APPLICATION USAGE**

42892 None.

42893 **RATIONALE**

42894 None.

42895 **FUTURE DIRECTIONS**

42896 None.

42897 **SEE ALSO**

42898 *exec, exit(), fork(), mlockall(), munmap()*

42899 XBD <sys/mman.h>

42900 **CHANGE HISTORY**

42901 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

42902 **Issue 6**

42903 The *mlock()* and *munlock()* functions are marked as part of the Range Memory Locking option.

42904 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 42905 implementation does not support the Range Memory Locking option.

NAME

mlockall, munlockall — lock/unlock the address space of a process (**REALTIME**)

SYNOPSIS

```
ML      #include <sys/mman.h>

int mlockall(int flags);
int munlockall(void);
```

DESCRIPTION

The *mlockall()* function shall cause all of the pages mapped by the address space of a process to be memory-resident until unlocked or until the process exits or *execs* another process image. The *flags* argument determines whether the pages to be locked are those currently mapped by the address space of the process, those that are mapped in the future, or both. The *flags* argument is constructed from the bitwise-inclusive OR of one or more of the following symbolic constants, defined in **<sys/mman.h>**:

MCL_CURRENT Lock all of the pages currently mapped into the address space of the process.

MCL_FUTURE Lock all of the pages that become mapped into the address space of the process in the future, when those mappings are established.

If **MCL_FUTURE** is specified, and the automatic locking of future mappings eventually causes the amount of locked memory to exceed the amount of available physical memory or any other implementation-defined limit, the behavior is implementation-defined. The manner in which the implementation informs the application of these situations is also implementation-defined.

The *munlockall()* function shall unlock all currently mapped pages of the address space of the process. Any pages that become mapped into the address space of the process after a call to *munlockall()* shall not be locked, unless there is an intervening call to *mlockall()* specifying **MCL_FUTURE** or a subsequent call to *mlockall()* specifying **MCL_CURRENT**. If pages mapped into the address space of the process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by a call by this process to *munlockall()*.

Upon successful return from the *mlockall()* function that specifies **MCL_CURRENT**, all currently mapped pages of the address space of the process shall be memory-resident and locked. Upon return from the *munlockall()* function, all currently mapped pages of the address space of the process shall be unlocked with respect to the address space of the process. The memory residency of unlocked pages is unspecified.

Appropriate privileges are required to lock process memory with *mlockall()*.

RETURN VALUE

Upon successful completion, the *mlockall()* function shall return a value of zero. Otherwise, no additional memory shall be locked, and the function shall return a value of **-1** and set *errno* to indicate the error. The effect of failure of *mlockall()* on previously existing locks in the address space is unspecified.

If it is supported by the implementation, the *munlockall()* function shall always return a value of zero. Otherwise, the function shall return a value of **-1** and set *errno* to indicate the error.

ERRORS

The *mlockall()* function shall fail if:

[EAGAIN] Some or all of the memory identified by the operation could not be locked when the call was made.

mlockall()*System Interfaces*

42950	[EINVAL]	The <i>flags</i> argument is zero, or includes unimplemented flags.
42951		The <i>mlockall()</i> function may fail if:
42952	[ENOMEM]	Locking all of the pages currently mapped into the address space of the process would exceed an implementation-defined limit on the amount of memory that the process may lock.
42953		
42954		
42955	[EPERM]	The calling process does not have appropriate privileges to perform the requested operation.
42956		
42957	EXAMPLES	
42958	None.	
42959	APPLICATION USAGE	
42960	None.	
42961	RATIONALE	
42962	None.	
42963	FUTURE DIRECTIONS	
42964	None.	
42965	SEE ALSO	
42966	<i>exec</i> , <i>exit()</i> , <i>fork()</i> , <i>mlock()</i> , <i>munmap()</i>	
42967	XBD < <i>sys/mman.h</i> >	
42968	CHANGE HISTORY	
42969	First released in Issue 5. Included for alignment with the POSIX Realtime Extension.	
42970	Issue 6	
42971	The <i>mlockall()</i> and <i>munlockall()</i> functions are marked as part of the Process Memory Locking option.	
42972		
42973	The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Memory Locking option.	
42974		

System Interfaces

mmap()**NAME**

mmap — map pages of memory

SYNOPSIS

```
#include <sys/mman.h>

void *mmap(void *addr, size_t len, int prot, int flags,
           int fildes, off_t off);
```

DESCRIPTION

The *mmap()* function shall establish a mapping between an address space of a process and a memory object.

The *mmap()* function shall be supported for the following memory objects:

- Regular files
- Shared memory objects
- Typed memory objects

Support for any other type of file is unspecified.

The format of the call is as follows:

```
pa=mmap(addr, len, prot, flags, fildes, off);
```

The *mmap()* function shall establish a mapping between the address space of the process at an address *pa* for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off* for *len* bytes. The value of *pa* is an implementation-defined function of the parameter *addr* and the values of *flags*, further described below. A successful *mmap()* call shall return *pa* as its result. The address range starting at *pa* and continuing for *len* bytes shall be legitimate for the possible (not necessarily current) address space of the process. The range of bytes starting at *off* and continuing for *len* bytes shall be legitimate for the possible (not necessarily current) offsets in the memory object represented by *fildes*.

If *fildes* represents a typed memory object opened with either the POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, the memory object to be mapped shall be that portion of the typed memory object allocated by the implementation as specified below. In this case, if *off* is non-zero, the behavior of *mmap()* is undefined. If *fildes* refers to a valid typed memory object that is not accessible from the calling process, *mmap()* shall fail.

The mapping established by *mmap()* shall replace any previous mappings for those whole pages containing any part of the address space of the process starting at *pa* and continuing for *len* bytes.

If the size of the mapped file changes after the call to *mmap()* as a result of some other operation on the mapped file, the effect of references to portions of the mapped region that correspond to added or removed portions of the file is unspecified.

If *len* is zero, *mmap()* shall fail and no mapping shall be established.

The parameter *prot* determines whether read, write, execute, or some combination of accesses are permitted to the data being mapped. The *prot* shall be either PROT_NONE or the bitwise-inclusive OR of one or more of the other flags in the following table, defined in the *<sys/mman.h>* header.

mmap()

System Interfaces

43016
43017
43018
43019
43020

Symbolic Constant	Description
PROT_READ	Data can be read.
PROT_WRITE	Data can be written.
PROT_EXEC	Data can be executed.
PROT_NONE	Data cannot be accessed.

43021 If an implementation cannot support the combination of access types specified by *prot*, the call to
43022 *mmap()* shall fail.

43023 An implementation may permit accesses other than those specified by *prot*; however, the
43024 implementation shall not permit a write to succeed where PROT_WRITE has not been set and
43025 shall not permit any access where PROT_NONE alone has been set. The implementation shall
43026 support at least the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and
43027 the bitwise-inclusive OR of PROT_READ and PROT_WRITE. The file descriptor *fildes* shall have
43028 been opened with read permission, regardless of the protection options specified. If
43029 PROT_WRITE is specified, the application shall ensure that it has opened the file descriptor *fildes*
43030 with write permission unless MAP_PRIVATE is specified in the *flags* parameter as described
43031 below.

43032 The parameter *flags* provides other information about the handling of the mapped data. The
43033 value of *flags* is the bitwise-inclusive OR of these options, defined in **<sys/mman.h>**:

43034
43035
43036
43037

Symbolic Constant	Description
MAP_SHARED	Changes are shared.
MAP_PRIVATE	Changes are private.
MAP_FIXED	Interpret <i>addr</i> exactly.

43038 XSI It is implementation-defined whether MAP_FIXED shall be supported. MAP_FIXED shall be
43039 supported on XSI-conformant systems.

43040 MAP_SHARED and MAP_PRIVATE describe the disposition of write references to the memory
43041 object. If MAP_SHARED is specified, write references shall change the underlying object. If
43042 MAP_PRIVATE is specified, modifications to the mapped data by the calling process shall be
43043 visible only to the calling process and shall not change the underlying object. It is unspecified
43044 whether modifications to the underlying object done after the MAP_PRIVATE mapping is
43045 established are visible through the MAP_PRIVATE mapping. Either MAP_SHARED or
43046 MAP_PRIVATE can be specified, but not both. The mapping type is retained across *fork()*.

43047 The state of synchronization objects such as mutexes, semaphores, barriers, and conditional
43048 variables placed in shared memory mapped with MAP_SHARED becomes undefined when the
43049 last region in any process containing the synchronization object is unmapped.

43050 TYM When *fildes* represents a typed memory object opened with either the
43051 POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG
43052 flag, *mmap()* shall, if there are enough resources available, map *len* bytes allocated from the
43053 corresponding typed memory object which were not previously allocated to any process in any
43054 processor that may access that typed memory object. If there are not enough resources available,
43055 the function shall fail. If *fildes* represents a typed memory object opened with the
43056 POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, these allocated bytes shall be contiguous
43057 within the typed memory object. If *fildes* represents a typed memory object opened with the
43058 POSIX_TYPED_MEM_ALLOCATE flag, these allocated bytes may be composed of non-
43059 contiguous fragments within the typed memory object. If *fildes* represents a typed memory
43060 object opened with neither the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag nor the
43061 POSIX_TYPED_MEM_ALLOCATE flag, *len* bytes starting at offset *off* within the typed memory
43062 object are mapped, exactly as when mapping a file or shared memory object. In this case, if two

processes map an area of typed memory using the same *off* and *len* values and using file descriptors that refer to the same memory pool (either from the same port or from a different port), both processes shall map the same region of storage.

When MAP_FIXED is set in the *flags* argument, the implementation is informed that the value of *pa* shall be *addr*, exactly. If MAP_FIXED is set, *mmap()* may return MAP_FAILED and set *errno* to [EINVAL]. If a MAP_FIXED request is successful, the mapping established by *mmap()* replaces any previous mappings for the pages in the range [*pa*,*pa*+*len*) of the process.

When MAP_FIXED is not set, the implementation uses *addr* in an implementation-defined manner to arrive at *pa*. The *pa* so chosen shall be an area of the address space that the implementation deems suitable for a mapping of *len* bytes to the file. All implementations interpret an *addr* value of 0 as granting the implementation complete freedom in selecting *pa*, subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a process address near which the mapping should be placed. When the implementation selects a value for *pa*, it never places a mapping at address 0, nor does it replace any extant mapping.

If MAP_FIXED is specified and *addr* is non-zero, it shall have the same remainder as the *off* parameter, modulo the page size as returned by *sysconf()* when passed _SC_PAGESIZE or _SC_PAGE_SIZE. The implementation may require that *off* is a multiple of the page size. If MAP_FIXED is specified, the implementation may require that *addr* is a multiple of the page size. The system performs mapping operations over whole pages. Thus, while the parameter *len* need not meet a size or alignment constraint, the system shall include, in any mapping operation, any partial page specified by the address range starting at *pa* and continuing for *len* bytes.

The system shall always zero-fill any partial page at the end of an object. Further, the system shall never write out any modified portions of the last page of an object which are beyond its end. References within the address range starting at *pa* and continuing for *len* bytes to whole pages following the end of an object shall result in delivery of a SIGBUS signal.

An implementation may generate SIGBUS signals when a reference would cause an error in the mapped object, such as out-of-space condition.

The *mmap()* function shall add an extra reference to the file associated with the file descriptor *fd* which is not removed by a subsequent *close()* on that file descriptor. This reference shall be removed when there are no more mappings to the file.

The last data access timestamp of the mapped file may be marked for update at any time between the *mmap()* call and the corresponding *munmap()* call. The initial read or write reference to a mapped region shall cause the file's last data access timestamp to be marked for update if it has not already been marked for update.

The last data modification and last file status change timestamps of a file that is mapped with MAP_SHARED and PROT_WRITE shall be marked for update at some point in the interval between a write reference to the mapped region and the next call to *msync()* with MS_ASYNC or MS_SYNC for that portion of the file by any process. If there is no such call and if the underlying file is modified as a result of a write reference, then these timestamps shall be marked for update at some time after the write reference.

There may be implementation-defined limits on the number of memory regions that can be mapped (per process or per system).

If such a limit is imposed, whether the number of memory regions that can be mapped by a process is decreased by the use of *shmat()* is implementation-defined.

If *mmap()* fails for reasons other than [EBADF], [EINVAL], or [ENOTSUP], some of the

mmap()*System Interfaces*

43109		mappings in the address range starting at <i>addr</i> and continuing for <i>len</i> bytes may have been
43110		unmapped.
43111	RETURN VALUE	
43112		Upon successful completion, the <i>mmap()</i> function shall return the address at which the mapping
43113		was placed (<i>pa</i>); otherwise, it shall return a value of MAP_FAILED and set <i>errno</i> to indicate the
43114		error. The symbol MAP_FAILED is defined in the <sys/mman.h> header. No successful return
43115		from <i>mmap()</i> shall return the value MAP_FAILED.
43116	ERRORS	
43117		The <i>mmap()</i> function shall fail if:
43118	[EACCES]	The <i>fildes</i> argument is not open for read, regardless of the protection specified,
43119		or <i>fildes</i> is not open for write and PROT_WRITE was specified for a
43120		MAP_SHARED type mapping.
43121	ML	[EAGAIN] The mapping could not be locked in memory, if required by <i>mlockall()</i> , due to
43122		a lack of resources.
43123		[EBADF] The <i>fildes</i> argument is not a valid open file descriptor.
43124		[EINVAL] The value of <i>len</i> is zero.
43125		[EINVAL] The value of <i>flags</i> is invalid (neither MAP_PRIVATE nor MAP_SHARED is
43126		set).
43127		[EMFILE] The number of mapped regions would exceed an implementation-defined
43128		limit (per process or per system).
43129		[ENODEV] The <i>fildes</i> argument refers to a file whose type is not supported by <i>mmap()</i> .
43130		[ENOMEM] MAP_FIXED was specified, and the range [<i>addr</i> , <i>addr</i> + <i>len</i>) exceeds that allowed
43131		for the address space of a process; or, if MAP_FIXED was not specified and
43132		there is insufficient room in the address space to effect the mapping.
43133	ML	[ENOMEM] The mapping could not be locked in memory, if required by <i>mlockall()</i> ,
43134		because it would require more space than the system is able to supply.
43135	TYM	[ENOMEM] Not enough unallocated memory resources remain in the typed memory
43136		object designated by <i>fildes</i> to allocate <i>len</i> bytes.
43137		[ENOTSUP] MAP_FIXED or MAP_PRIVATE was specified in the <i>flags</i> argument and the
43138		implementation does not support this functionality.
43139		The implementation does not support the combination of accesses requested
43140		in the <i>prot</i> argument.
43141		[ENXIO] Addresses in the range [<i>off</i> , <i>off</i> + <i>len</i>) are invalid for the object specified by <i>fildes</i> .
43142		[ENXIO] MAP_FIXED was specified in <i>flags</i> and the combination of <i>addr</i> , <i>len</i> , and <i>off</i> is
43143		invalid for the object specified by <i>fildes</i> .
43144	TYM	[ENXIO] The <i>fildes</i> argument refers to a typed memory object that is not accessible from
43145		the calling process.
43146		[EOVERFLOW] The file is a regular file and the value of <i>off</i> plus <i>len</i> exceeds the offset
43147		maximum established in the open file description associated with <i>fildes</i> .

43148 The *mmap()* function may fail if:

43149 [EINVAL] The *addr* argument (if MAP_FIXED was specified) or *off* is not a multiple of the
 43150 page size as returned by *sysconf()*, or is considered invalid by the
 43151 implementation.

43152 EXAMPLES

43153 None.

43154 APPLICATION USAGE

43155 Use of *mmap()* may reduce the amount of memory available to other memory allocation
 43156 functions.

43157 Use of MAP_FIXED may result in unspecified behavior in further use of *malloc()* and *shmat()*.
 43158 The use of MAP_FIXED is discouraged, as it may prevent an implementation from making the
 43159 most effective use of resources. Most implementations require that *off* and *addr* are multiples of
 43160 the page size as returned by *sysconf()*.

43161 The application must ensure correct synchronization when using *mmap()* in conjunction with
 43162 any other file access method, such as *read()* and *write()*, standard input/output, and *shmat()*.

43163 The *mmap()* function allows access to resources via address space manipulations, instead of
 43164 *read()/write()*. Once a file is mapped, all a process has to do to access it is use the data at the
 43165 address to which the file was mapped. So, using pseudo-code to illustrate the way in which an
 43166 existing program might be changed to use *mmap()*, the following:

```
43167 fildes = open(...)
43168 lseek(fildes, some_offset)
43169 read(fildes, buf, len)
43170 /* Use data in buf. */
```

43171 becomes:

```
43172 fildes = open(...)
43173 address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
43174 /* Use data at address. */
```

43175 RATIONALE

43176 After considering several other alternatives, it was decided to adopt the *mmap()* definition
 43177 found in SVR4 for mapping memory objects into process address spaces. The SVR4 definition is
 43178 minimal, in that it describes only what has been built, and what appears to be necessary for a
 43179 general and portable mapping facility.

43180 Note that while *mmap()* was first designed for mapping files, it is actually a general-purpose
 43181 mapping facility. It can be used to map any appropriate object, such as memory, files, devices,
 43182 and so on, into the address space of a process.

43183 When a mapping is established, it is possible that the implementation may need to map more
 43184 than is requested into the address space of the process because of hardware requirements. An
 43185 application, however, cannot count on this behavior. Implementations that do not use a paged
 43186 architecture may simply allocate a common memory region and return the address of it; such
 43187 implementations probably do not allocate any more than is necessary. References past the end of
 43188 the requested area are unspecified.

43189 If an application requests a mapping that would overlay existing mappings in the process, it
 43190 might be desirable that an implementation detect this and inform the application. However, the
 43191 default, portable (not MAP_FIXED) operation does not overlay existing mappings. On the other
 43192 hand, if the program specifies a fixed address mapping (which requires some implementation

mmap()*System Interfaces*

knowledge to determine a suitable address, if the function is supported at all), then the program is presumed to be successfully managing its own address space and should be trusted when it asks to map over existing data structures. Furthermore, it is also desirable to make as few system calls as possible, and it might be considered onerous to require an *munmap()* before an *mmap()* to the same address range. This volume of POSIX.1-2008 specifies that the new mappings replace any existing mappings, following existing practice in this regard.

It is not expected that all hardware implementations are able to support all combinations of permissions at all addresses. Implementations are required to disallow write access to mappings without write permission and to disallow access to mappings without any access permission. Other than these restrictions, implementations may allow access types other than those requested by the application. For example, if the application requests only `PROT_WRITE`, the implementation may also allow read access. A call to *mmap()* fails if the implementation cannot support allowing all the access requested by the application. For example, some implementations cannot support a request for both write access and execute access simultaneously. All implementations must support requests for no access, read access, write access, and both read and write access. Strictly conforming code must only rely on the required checks. These restrictions allow for portability across a wide range of hardware.

The `MAP_FIXED` address treatment is likely to fail for non-page-aligned values and for certain architecture-dependent address ranges. Conforming implementations cannot count on being able to choose address values for `MAP_FIXED` without utilizing non-portable, implementation-defined knowledge. Nonetheless, `MAP_FIXED` is provided as a standard interface conforming to existing practice for utilizing such knowledge when it is available.

Similarly, in order to allow implementations that do not support virtual addresses, support for directly specifying any mapping addresses via `MAP_FIXED` is not required and thus a conforming application may not count on it.

The `MAP_PRIVATE` function can be implemented efficiently when memory protection hardware is available. When such hardware is not available, implementations can implement such “mappings” by simply making a real copy of the relevant data into process private memory, though this tends to behave similarly to *read()*.

The function has been defined to allow for many different models of using shared memory. However, all uses are not equally portable across all machine architectures. In particular, the *mmap()* function allows the system as well as the application to specify the address at which to map a specific region of a memory object. The most portable way to use the function is always to let the system choose the address, specifying `NULL` as the value for the argument *addr* and not to specify `MAP_FIXED`.

If it is intended that a particular region of a memory object be mapped at the same address in a group of processes (on machines where this is even possible), then `MAP_FIXED` can be used to pass in the desired mapping address. The system can still be used to choose the desired address if the first such mapping is made without specifying `MAP_FIXED`, and then the resulting mapping address can be passed to subsequent processes for them to pass in via `MAP_FIXED`. The availability of a specific address range cannot be guaranteed, in general.

The *mmap()* function can be used to map a region of memory that is larger than the current size of the object. Memory access within the mapping but beyond the current end of the underlying objects may result in `SIGBUS` signals being sent to the process. The reason for this is that the size of the object can be manipulated by other processes and can change at any moment. The implementation should tell the application that a memory reference is outside the object where this can be detected; otherwise, written data may be lost and read data may not reflect actual data in the object.

Note that references beyond the end of the object do not extend the object as the new end cannot be determined precisely by most virtual memory hardware. Instead, the size can be directly manipulated by *ftruncate()*.

Process memory locking does apply to shared memory regions, and the MEMLOCK_FUTURE argument to *mlockall()* can be relied upon to cause new shared memory regions to be automatically locked.

Existing implementations of *mmap()* return the value `-1` when unsuccessful. Since the casting of this value to type **void *** cannot be guaranteed by the ISO C standard to be distinct from a successful value, this volume of POSIX.1-2008 defines the symbol MAP_FAILED, which a conforming implementation does not return as the result of a successful call.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fcntl()*, *fork()*, *lockf()*, *msync()*, *munmap()*, *mprotect()*, *posix_typed_mem_open()*, *shmat()*, *sysconf()*

XBD <sys/mman.h>

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Aligned with *mmap()* in the POSIX Realtime Extension as follows:

- The DESCRIPTION is extensively reworded.
- The [EAGAIN] and [ENOTSUP] mandatory error conditions are added.
- New cases of [ENOMEM] and [ENXIO] are added as mandatory error conditions.
- The value returned on failure is the value of the constant MAP_FAILED; this was previously defined as `-1`.

Large File Summit extensions are added.

Issue 6

The *mmap()* function is marked as part of the Memory Mapped Files option.

The Open Group Corrigendum U028/6 is applied, changing (**void ***)`-1` to MAP_FAILED.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to describe the use of MAP_FIXED.
- The DESCRIPTION is updated to describe the addition of an extra reference to the file associated with the file descriptor passed to *mmap()*.
- The DESCRIPTION is updated to state that there may be implementation-defined limits on the number of memory regions that can be mapped.
- The DESCRIPTION is updated to describe constraints on the alignment and size of the *off* argument.

mmap()*System Interfaces*

- 43280 • The [EINVAL] and [EMFILE] error conditions are added.
- 43281 • The [EOVERFLOW] error condition is added. This change is to support large files.
- 43282 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
- 43283 • The DESCRIPTION is updated to describe the cases when MAP_PRIVATE and
- 43284 MAP_FIXED need not be supported.
- 43285 The following changes are made for alignment with IEEE Std 1003.1j-2000:
- 43286 • Semantics for typed memory objects are added to the DESCRIPTION.
- 43287 • New [ENOMEM] and [ENXIO] errors are added to the ERRORS section.
- 43288 • The *posix_typed_mem_open()* function is added to the SEE ALSO section.
- 43289 The normative text is updated to avoid use of the term “must” for application requirements.
- 43290 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code
- 43291 in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM).
- 43292 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/60 is applied, updating the
- 43293 DESCRIPTION and ERRORS sections to add the [EINVAL] error when *len* is zero.
- 43294 **Issue 7**
- 43295 Austin Group Interpretations 1003.1-2001 #078 and #079 are applied, clarifying page alignment
- 43296 requirements and adding a note about the state of synchronization objects becoming undefined
- 43297 when a shared region is unmapped.
- 43298 Functionality relating to the Memory Protection and Memory Mapped Files options is moved to
- 43299 the Base.
- 43300 Changes are made related to support for finegrained timestamps.

NAME

modf, modff, modfl — decompose a floating-point number

SYNOPSIS

```
#include <math.h>

double modf(double x, double *iptr);
float modff(float value, float *iptr);
long double modfl(long double value, long double *iptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall break the argument *x* into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a **double** (for the *modf()* function), a **float** (for the *modff()* function), or a **long double** (for the *modfl()* function), in the object pointed to by *iptr*.

RETURN VALUE

Upon successful completion, these functions shall return the signed fractional part of *x*.

MX If *x* is NaN, a NaN shall be returned, and **iptr* shall be set to a NaN.

If *x* is $\pm\text{Inf}$, ± 0 shall be returned, and **iptr* shall be set to $\pm\text{Inf}$.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *modf()* function computes the function result and **iptr* such that:

```
a = modf(x, iptr) ;
x == a+*iptr ;
```

allowing for the usual floating-point inaccuracies.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

frexp(), *isnan()*, *ldexp()*

XBD **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

modf()*System Interfaces*43341 **Issue 6**

43342 The *modff()* and *modfl()* functions are added for alignment with the ISO/IEC 9899:1999
43343 standard.

43344 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
43345 revised to align with the ISO/IEC 9899:1999 standard.

43346 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
43347 marked.

43348 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/35 is applied, correcting the code example
43349 in the APPLICATION USAGE section.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

mprotect — set protection of memory mapping

SYNOPSIS

```
#include <sys/mman.h>
```

```
int mprotect(void *addr, size_t len, int prot);
```

DESCRIPTION

The *mprotect()* function shall change the access protections to be that specified by *prot* for those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes. The parameter *prot* determines whether read, write, execute, or some combination of accesses are permitted to the data being mapped. The *prot* argument should be either *PROT_NONE* or the bitwise-inclusive OR of one or more of *PROT_READ*, *PROT_WRITE*, and *PROT_EXEC*.

If an implementation cannot support the combination of access types specified by *prot*, the call to *mprotect()* shall fail.

An implementation may permit accesses other than those specified by *prot*; however, no implementation shall permit a write to succeed where *PROT_WRITE* has not been set or shall permit any access where *PROT_NONE* alone has been set. Implementations shall support at least the following values of *prot*: *PROT_NONE*, *PROT_READ*, *PROT_WRITE*, and the bitwise-inclusive OR of *PROT_READ* and *PROT_WRITE*. If *PROT_WRITE* is specified, the application shall ensure that it has opened the mapped objects in the specified address range with write permission, unless *MAP_PRIVATE* was specified in the original mapping, regardless of whether the file descriptors used to map the objects have since been closed.

The implementation may require that *addr* be a multiple of the page size as returned by *sysconf()*.

The behavior of this function is unspecified if the mapping was not established by a call to *mmap()*.

When *mprotect()* fails for reasons other than *[EINVAL]*, the protections on some of the pages in the range *[addr,addr+len)* may have been changed.

RETURN VALUE

Upon successful completion, *mprotect()* shall return 0; otherwise, it shall return *-1* and set *errno* to indicate the error.

ERRORS

The *mprotect()* function shall fail if:

[EACCESS]	The <i>prot</i> argument specifies a protection that violates the access permission the process has to the underlying memory object.
[EAGAIN]	The <i>prot</i> argument specifies <i>PROT_WRITE</i> over a <i>MAP_PRIVATE</i> mapping and there are insufficient memory resources to reserve for locking the private page.
[ENOMEM]	Addresses in the range <i>[addr,addr+len)</i> are invalid for the address space of a process, or specify one or more pages which are not mapped.
[ENOMEM]	The <i>prot</i> argument specifies <i>PROT_WRITE</i> on a <i>MAP_PRIVATE</i> mapping, and it would require more space than the system is able to supply for locking the private pages, if required.

mprotect()*System Interfaces*

- 43393 [ENOTSUP] The implementation does not support the combination of accesses requested
43394 in the *prot* argument.
- 43395 The *mprotect()* function may fail if:
- 43396 [EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf()*.
- 43397 **EXAMPLES**
- 43398 None.
- 43399 **APPLICATION USAGE**
- 43400 Most implementations require that *addr* is a multiple of the page size as returned by *sysconf()*.
- 43401 **RATIONALE**
- 43402 None.
- 43403 **FUTURE DIRECTIONS**
- 43404 None.
- 43405 **SEE ALSO**
- 43406 *mmap()*, *sysconf()*
- 43407 XBD <sys/mman.h>
- 43408 **CHANGE HISTORY**
- 43409 First released in Issue 4, Version 2.
- 43410 **Issue 5**
- 43411 Moved from X/OPEN UNIX extension to BASE.
- 43412 Aligned with *mprotect()* in the POSIX Realtime Extension as follows:
- 43413 • The DESCRIPTION is largely reworded.
- 43414 • [ENOTSUP] and a second form of [ENOMEM] are added as mandatory error conditions.
- 43415 • [EAGAIN] is moved from the optional to the mandatory error conditions.
- 43416 **Issue 6**
- 43417 The *mprotect()* function is marked as part of the Memory Protection option.
- 43418 The following new requirements on POSIX implementations derive from alignment with the
- 43419 Single UNIX Specification:
- 43420 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple
- 43421 of the page size as returned by *sysconf()*.
- 43422 • The [EINVAL] error condition is added.
- 43423 The normative text is updated to avoid use of the term “must” for application requirements.
- 43424 **Issue 7**
- 43425 SD5-XSH-ERN-22 is applied, deleting erroneous APPLICATION USAGE.
- 43426 Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment
- 43427 requirements.
- 43428 The *mprotect()* function is moved from the Memory Protection option to the Base.

NAMEmq_close — close a message queue (**REALTIME**)**SYNOPSIS**

```
#include <mqueue.h>

int mq_close(mqd_t mqdes);
```

DESCRIPTION

The *mq_close()* function shall remove the association between the message queue descriptor, *mqdes*, and its message queue. The results of using this message queue descriptor after successful return from this *mq_close()*, and until the return of this message queue descriptor from a subsequent *mq_open()*, are undefined.

If the process has successfully attached a notification request to the message queue via this *mqdes*, this attachment shall be removed, and the message queue is available for another process to attach for notification.

RETURN VALUE

Upon successful completion, the *mq_close()* function shall return a value of zero; otherwise, the function shall return a value of -1 and set *errno* to indicate the error.

ERRORS

The *mq_close()* function shall fail if:

[EBADF] The *mqdes* argument is not a valid message queue descriptor.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open(), *mq_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

XBD **<mqueue.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_close()* function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

mq_getattr()

System Interfaces

43465 **NAME**43466 **mq_getattr** — get message queue attributes (**REALTIME**)43467 **SYNOPSIS**

```
43468 MSG      #include <mqqueue.h>
43469          int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

43470 **DESCRIPTION**

43471 The *mq_getattr()* function shall obtain status information and attributes of the message queue
 43472 and the open message queue description associated with the message queue descriptor.

43473 The *mqdes* argument specifies a message queue descriptor.

43474 The results shall be returned in the **mq_attr** structure referenced by the *mqstat* argument.

43475 Upon return, the following members shall have the values associated with the open message
 43476 queue description as set when the message queue was opened and as modified by subsequent
 43477 *mq_setattr()* calls: *mq_flags*.

43478 The following attributes of the message queue shall be returned as set at message queue
 43479 creation: *mq_maxmsg*, *mq_msgsize*.

43480 Upon return, the following members within the **mq_attr** structure referenced by the *mqstat*
 43481 argument shall be set to the current state of the message queue:

43482 *mq_curmsgs* The number of messages currently on the queue.

43483 **RETURN VALUE**

43484 Upon successful completion, the *mq_getattr()* function shall return zero. Otherwise, the function
 43485 shall return -1 and set *errno* to indicate the error.

43486 **ERRORS**

43487 The *mq_getattr()* function may fail if:

43488 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

43489 **EXAMPLES**

43490 See *mq_notify()*.

43491 **APPLICATION USAGE**

43492 None.

43493 **RATIONALE**

43494 None.

43495 **FUTURE DIRECTIONS**

43496 None.

43497 **SEE ALSO**

43498 *mq_notify()*, *mq_open()*, *mq_send()*, *mq_setattr()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

43499 XBD **<mqqueue.h>**

43500 **CHANGE HISTORY**

43501 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

43502 **Issue 6**

43503 The *mq_getattr()* function is marked as part of the Message Passing option.

43504 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 43505 implementation does not support the Message Passing option.

- 43506 The *mq_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std
43507 1003.1d-1999.
- 43508 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/61 is applied, updating the ERRORS
43509 section to change the [EBADF] error from mandatory to optional.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

mq_notify()*System Interfaces*43510 **NAME**43511 mq_notify — notify process that a message is available (**REALTIME**)43512 **SYNOPSIS**

```
43513 MSG      #include <mqueue.h>
43514          int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```

43515 **DESCRIPTION**

43516 If the argument *notification* is not NULL, this function shall register the calling process to be
 43517 notified of message arrival at an empty message queue associated with the specified message
 43518 queue descriptor, *mqdes*. The notification specified by the *notification* argument shall be sent to
 43519 the process when the message queue transitions from empty to non-empty. At any time, only
 43520 one process may be registered for notification by a message queue. If the calling process or any
 43521 other process has already registered for notification of message arrival at the specified message
 43522 queue, subsequent attempts to register for that message queue shall fail.

43523 If *notification* is NULL and the process is currently registered for notification by the specified
 43524 message queue, the existing registration shall be removed.

43525 When the notification is sent to the registered process, its registration shall be removed. The
 43526 message queue shall then be available for registration.

43527 If a process has registered for notification of message arrival at a message queue and some
 43528 thread is blocked in *mq_receive()* or *mq_timedreceive()* waiting to receive a message when a
 43529 message arrives at the queue, the arriving message shall satisfy the appropriate *mq_receive()* or
 43530 *mq_timedreceive()*, respectively. The resulting behavior is as if the message queue remains empty,
 43531 and no notification shall be sent.

43532 **RETURN VALUE**

43533 Upon successful completion, the *mq_notify()* function shall return a value of zero; otherwise, the
 43534 function shall return a value of -1 and set *errno* to indicate the error.

43535 **ERRORS**

43536 The *mq_notify()* function shall fail if:

43537 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

43538 [EBUSY] A process is already registered for notification by the message queue.

43539 The *mq_notify()* function may fail if:

43540 [EINVAL] The *notification* argument is NULL and the process is currently not registered.

43541 **EXAMPLES**

43542 The following program registers a notification request for the message queue named in its
 43543 command-line argument. Notification is performed by creating a thread. The thread executes a
 43544 function which reads one message from the queue and then terminates the process.

```
43545 #include <pthread.h>
43546 #include <mqueue.h>
43547 #include <assert.h>
43548 #include <stdio.h>
43549 #include <stdlib.h>
43550 #include <unistd.h>

43551 static void                /* Thread start function */
43552 tfunc(union sigval sv)
43553 {
```


System Interfaces

mq_notify()

```

43554     struct mq_attr attr;
43555     ssize_t nr;
43556     void *buf;
43557     mqd_t mqdes = *((mqd_t *) sv.sival_ptr);
43558     /* Determine maximum msg size; allocate buffer to receive msg */
43559     if (mq_getattr(mqdes, &attr) == -1) {
43560         perror("mq_getattr");
43561         exit(EXIT_FAILURE);
43562     }
43563     buf = malloc(attr.mq_msgsize);
43564     if (buf == NULL) {
43565         perror("malloc");
43566         exit(EXIT_FAILURE);
43567     }
43568     nr = mq_receive(mqdes, buf, attr.mq_msgsize, NULL);
43569     if (nr == -1) {
43570         perror("mq_receive");
43571         exit(EXIT_FAILURE);
43572     }
43573     printf("Read %ld bytes from message queue\n", (long) nr);
43574     free(buf);
43575     exit(EXIT_SUCCESS);          /* Terminate the process */
43576 }
43577
43578 int
43579 main(int argc, char *argv[])
43580 {
43581     mqd_t mqdes;
43582     struct sigevent not;
43583     assert(argc == 2);
43584     mqdes = mq_open(argv[1], O_RDONLY);
43585     if (mqdes == (mqd_t) -1) {
43586         perror("mq_open");
43587         exit(EXIT_FAILURE);
43588     }
43589     not.sigev_notify = SIGEV_THREAD;
43590     not.sigev_notify_function = tfunc;
43591     not.sigev_notify_attributes = NULL;
43592     not.sigev_value.sival_ptr = &mqdes; /* Arg. to thread func. */
43593     if (mq_notify(mqdes, &not) == -1) {
43594         perror("mq_notify");
43595         exit(EXIT_FAILURE);
43596     }
43597     pause(); /* Process will be terminated by thread function */
43598 }

```

mq_notify()

System Interfaces

43598 **APPLICATION USAGE**

43599 None.

43600 **RATIONALE**

43601 None.

43602 **FUTURE DIRECTIONS**

43603 None.

43604 **SEE ALSO**43605 *mq_open()*, *mq_send()*, *mq_receive()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*43606 XBD **<mqqueue.h>**43607 **CHANGE HISTORY**

43608 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

43609 **Issue 6**43610 The *mq_notify()* function is marked as part of the Message Passing option.43611 The [ENOSYS] error condition has been removed as stubs need not be provided if an
43612 implementation does not support the Message Passing option.43613 The *mq_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std
43614 1003.1d-1999.43615 **Issue 7**43616 SD5-XSH-ERN-38 is applied, adding the *mq_timedreceive()* function to the DESCRIPTION.

43617 Austin Group Interpretation 1003.1-2001 #032 is applied, adding the [EINVAL] error.

43618 An example is added.

NAME

mq_open — open a message queue (REALTIME)

SYNOPSIS

```
#include <mqueue.h>

mqd_t mq_open(const char *name, int oflag, ...);
```

DESCRIPTION

The *mq_open()* function shall establish the connection between a process and a message queue with a message queue descriptor. It shall create an open message queue description that refers to the message queue, and a message queue descriptor that refers to that open message queue description. The message queue descriptor is used by other functions to refer to that message queue. The *name* argument points to a string naming a message queue. It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The *name* argument conforms to the construction rules for a pathname, except that the interpretation of <slash> characters other than the leading <slash> character in *name* is implementation-defined, and that the length limits for the *name* argument are implementation-defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If *name* begins with the <slash> character, then processes calling *mq_open()* with the same value of *name* shall refer to the same message queue object, as long as that name has not been removed. If *name* does not begin with the <slash> character, the effect is implementation-defined. If the *name* argument is not the name of an existing message queue and creation is not requested, *mq_open()* shall fail and return an error.

A message queue descriptor may be implemented using a file descriptor, in which case applications can open up to at least {OPEN_MAX} file and message queues.

The *oflag* argument requests the desired receive and/or send access to the message queue. The requested access permission to receive messages or send messages shall be granted if the calling process would be granted read or write access, respectively, to an equivalently protected file.

The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications shall specify exactly one of the first three values (access modes) below in the value of *oflag*:

O_RDONLY Open the message queue for receiving messages. The process can use the returned message queue descriptor with *mq_receive()*, but not *mq_send()*. A message queue may be open multiple times in the same or different processes for receiving messages.

O_WRONLY Open the queue for sending messages. The process can use the returned message queue descriptor with *mq_send()* but not *mq_receive()*. A message queue may be open multiple times in the same or different processes for sending messages.

O_RDWR Open the queue for both receiving and sending messages. The process can use any of the functions allowed for **O_RDONLY** and **O_WRONLY**. A message queue may be open multiple times in the same or different processes for sending messages.

Any combination of the remaining flags may be specified in the value of *oflag*:

O_CREAT Create a message queue. It requires two additional arguments: *mode*, which shall be of type **mode_t**, and *attr*, which shall be a pointer to an **mq_attr** structure. If the pathname *name* has already been used to create a message queue that still exists, then this flag shall have no effect, except as noted under **O_EXCL**. Otherwise, a message queue shall be created without any messages

mq_open()*System Interfaces*

43665		in it. The user ID of the message queue shall be set to the effective user ID of
43666		the process. The group ID of the message queue shall be set to the effective
43667		group ID of the process; however, if the <i>name</i> argument is visible in the file
43668		system, the group ID may be set to the group ID of the containing directory.
43669		When bits in <i>mode</i> other than the file permission bits are specified, the effect is
43670		unspecified. If <i>attr</i> is NULL, the message queue shall be created with
43671		implementation-defined default message queue attributes. If <i>attr</i> is non-NULL
43672		and the calling process has appropriate privileges on <i>name</i> , the message queue
43673		<i>mq_maxmsg</i> and <i>mq_msgsize</i> attributes shall be set to the values of the
43674		corresponding members in the mq_attr structure referred to by <i>attr</i> . If <i>attr</i> is
43675		non-NULL, but the calling process does not have appropriate privileges on
43676		<i>name</i> , the <i>mq_open()</i> function shall fail and return an error without creating
43677		the message queue.
43678	O_EXCL	If O_EXCL and O_CREAT are set, <i>mq_open()</i> shall fail if the message queue
43679		<i>name</i> exists. The check for the existence of the message queue and the creation
43680		of the message queue if it does not exist shall be atomic with respect to other
43681		threads executing <i>mq_open()</i> naming the same <i>name</i> with O_EXCL and
43682		O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is
43683		undefined.
43684	O_NONBLOCK	Determines whether an <i>mq_send()</i> or <i>mq_receive()</i> waits for resources or
43685		messages that are not currently available, or fails with <i>errno</i> set to [EAGAIN];
43686		see <i>mq_send()</i> and <i>mq_receive()</i> for details.
43687		The <i>mq_open()</i> function does not add or remove messages from the queue.
43688	RETURN VALUE	
43689		Upon successful completion, the function shall return a message queue descriptor; otherwise,
43690		the function shall return (mqd_t)−1 and set <i>errno</i> to indicate the error.
43691	ERRORS	
43692		The <i>mq_open()</i> function shall fail if:
43693	[EACCES]	The message queue exists and the permissions specified by <i>oflag</i> are denied, or
43694		the message queue does not exist and permission to create the message queue
43695		is denied.
43696	[EEXIST]	O_CREAT and O_EXCL are set and the named message queue already exists.
43697	[EINTR]	The <i>mq_open()</i> function was interrupted by a signal.
43698	[EINVAL]	The <i>mq_open()</i> function is not supported for the given name.
43699	[EINVAL]	O_CREAT was specified in <i>oflag</i> , the value of <i>attr</i> is not NULL, and either
43700		<i>mq_maxmsg</i> or <i>mq_msgsize</i> was less than or equal to zero.
43701	[EMFILE]	Too many message queue descriptors or file descriptors are currently in use by
43702		this process.
43703	[ENFILE]	Too many message queues are currently open in the system.
43704	[ENOENT]	O_CREAT is not set and the named message queue does not exist.
43705	[ENOSPC]	There is insufficient space for the creation of the new message queue.

43706 If any of the following conditions occur, the *mq_open()* function may return (**mqd_t**)-1 and set
 43707 *errno* to the corresponding value.

43708 [ENAMETOOLONG]

43709 The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems
 43710 XSI that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI
 43711 systems, or has a pathname component that is longer than
 43712 XSI {_POSIX_NAME_MAX} on systems that do not support the XSI option or
 43713 longer than {_XOPEN_NAME_MAX} on XSI systems.

43714 EXAMPLES

43715 None.

43716 APPLICATION USAGE

43717 None.

43718 RATIONALE

43719 None.

43720 FUTURE DIRECTIONS

43721 A future version might require the *mq_open()* and *mq_unlink()* functions to have semantics
 43722 similar to normal file system operations.

43723 SEE ALSO

43724 *mq_close()*, *mq_getattr()*, *mq_receive()*, *mq_send()*, *mq_setattr()*, *mq_unlink()*, *msgctl()*, *msgget()*,
 43725 *msgrcv()*, *msgsnd()*

43726 XBD <mqqueue.h>

43727 CHANGE HISTORY

43728 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

43729 Issue 6

43730 The *mq_open()* function is marked as part of the Message Passing option.

43731 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 43732 implementation does not support the Message Passing option.

43733 The *mq_timedreceive()* and *mq_timedsend()* functions are added to the SEE ALSO section for
 43734 alignment with IEEE Std 1003.1d-1999.

43735 The DESCRIPTION of O_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

43736 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/62 is applied, updating the description of
 43737 the permission bits in the DESCRIPTION. The change is made for consistency with the
 43738 *shm_open()* and *sem_open()* functions.

43739 Issue 7

43740 Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and
 43741 changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

43742 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

43743 SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the
 43744 user ID and group ID of the message queue.

mq_receive()*System Interfaces*43745 **NAME**43746 **mq_receive, mq_timedreceive** — receive a message from a message queue (**REALTIME**)43747 **SYNOPSIS**

```

43748 MSG #include <mqueue.h>
43749
43749 ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
43750 unsigned *msg_prio);
43751
43751 #include <mqueue.h>
43752 #include <time.h>
43753
43753 ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
43754 size_t msg_len, unsigned *restrict msg_prio,
43755 const struct timespec *restrict abstime);

```

43756 **DESCRIPTION**

43757 The *mq_receive()* function shall receive the oldest of the highest priority message(s) from the
 43758 message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msg_len*
 43759 argument, is less than the *mq_msgsize* attribute of the message queue, the function shall fail and
 43760 return an error. Otherwise, the selected message shall be removed from the queue and copied to
 43761 the buffer pointed to by the *msg_ptr* argument.

43762 If the value of *msg_len* is greater than {SSIZE_MAX}, the result is implementation-defined.

43763 If the argument *msg_prio* is not NULL, the priority of the selected message shall be stored in the
 43764 location referenced by *msg_prio*.

43765 If the specified message queue is empty and O_NONBLOCK is not set in the message queue
 43766 description associated with *mqdes*, *mq_receive()* shall block until a message is enqueued on the
 43767 message queue or until *mq_receive()* is interrupted by a signal. If more than one thread is waiting
 43768 to receive a message when a message arrives at an empty queue and the Priority Scheduling
 43769 option is supported, then the thread of highest priority that has been waiting the longest shall be
 43770 selected to receive the message. Otherwise, it is unspecified which waiting thread receives the
 43771 message. If the specified message queue is empty and O_NONBLOCK is set in the message
 43772 queue description associated with *mqdes*, no message shall be removed from the queue, and
 43773 *mq_receive()* shall return an error.

43774 The *mq_timedreceive()* function shall receive the oldest of the highest priority messages from the
 43775 message queue specified by *mqdes* as described for the *mq_receive()* function. However, if
 43776 O_NONBLOCK was not specified when the message queue was opened via the *mq_open()*
 43777 function, and no message exists on the queue to satisfy the receive, the wait for such a message
 43778 shall be terminated when the specified timeout expires. If O_NONBLOCK is set, this function is
 43779 equivalent to *mq_receive()*.

43780 The timeout expires when the absolute time specified by *abstime* passes, as measured by the
 43781 clock on which timeouts are based (that is, when the value of that clock equals or exceeds
 43782 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the
 43783 call.

43784 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
 43785 be the resolution of the clock on which it is based. The *timespec* argument is defined in the
 43786 **<time.h>** header.

43787 Under no circumstance shall the operation fail with a timeout if a message can be removed from
 43788 the message queue immediately. The validity of the *abstime* parameter need not be checked if a
 43789 message can be removed from the message queue immediately.

RETURN VALUE

Upon successful completion, the *mq_receive()* and *mq_timedreceive()* functions shall return the length of the selected message in bytes and the message shall be removed from the queue. Otherwise, no message shall be removed from the queue, the functions shall return a value of -1 , and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

- [EAGAIN] O_NONBLOCK was set in the message description associated with *mqdes*, and the specified message queue is empty.
- [EBADF] The *mqdes* argument is not a valid message queue descriptor open for reading.
- [EMSGSIZE] The specified message buffer size, *msg_len*, is less than the message size attribute of the message queue.
- [EINTR] The *mq_receive()* or *mq_timedreceive()* operation was interrupted by a signal.
- [EINVAL] The process or thread would have blocked, and the *abstime* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.
- [ETIMEDOUT] The O_NONBLOCK flag was not set when the message queue was opened, but no message arrived on the queue before the specified timeout expired.

These functions may fail if:

- [EBADMSG] The implementation has detected a data corruption problem with the message.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open(), *mq_send()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, *time()*

XBD *<mqqueue.h>*, *<time.h>*

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_receive()* function is marked as part of the Message Passing option.

The Open Group Corrigendum U021/4 is applied. The DESCRIPTION is changed to refer to *msg_len* rather than *maxsize*.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

mq_receive()*System Interfaces*

- 43832 • In this function it is possible for the return value to exceed the range of the type **ssize_t**
43833 (since **size_t** has a larger range of positive values than **ssize_t**). A sentence restricting the
43834 size of the **size_t** object is added to the description to resolve this conflict.
- 43835 The *mq_timedreceive()* function is added for alignment with IEEE Std 1003.1d-1999.
- 43836 The **restrict** keyword is added to the *mq_timedreceive()* prototype for alignment with the
43837 ISO/IEC 9899:1999 standard.
- 43838 IEEE PASC Interpretation 1003.1 #109 is applied, correcting the return type for *mq_timedreceive()*
43839 from **int** to **ssize_t**.
- 43840 **Issue 7**
- 43841 The *mq_timedreceive()* function is moved from the Timeouts option to the Base.
- 43842 Functionality relating to the Timers option is moved to the Base.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

mq_send()**NAME**

`mq_send`, `mq_timedsend` — send a message to a message queue (**REALTIME**)

SYNOPSIS

```
#include <mqueue.h>

int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
            unsigned msg_prio);

#include <mqueue.h>
#include <time.h>

int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
                unsigned msg_prio, const struct timespec *abstime);
```

DESCRIPTION

The `mq_send()` function shall add the message pointed to by the argument `msg_ptr` to the message queue specified by `mqdes`. The `msg_len` argument specifies the length of the message, in bytes, pointed to by `msg_ptr`. The value of `msg_len` shall be less than or equal to the `mq_msgsize` attribute of the message queue, or `mq_send()` shall fail.

If the specified message queue is not full, `mq_send()` shall behave as if the message is inserted into the message queue at the position indicated by the `msg_prio` argument. A message with a larger numeric value of `msg_prio` shall be inserted before messages with lower values of `msg_prio`. A message shall be inserted after other messages in the queue, if any, with equal `msg_prio`. The value of `msg_prio` shall be less than `[MQ_PRIO_MAX]`.

If the specified message queue is full and `O_NONBLOCK` is not set in the message queue description associated with `mqdes`, `mq_send()` shall block until space becomes available to enqueue the message, or until `mq_send()` is interrupted by a signal. If more than one thread is waiting to send when space becomes available in the message queue and the Priority Scheduling option is supported, then the thread of the highest priority that has been waiting the longest shall be unblocked to send its message. Otherwise, it is unspecified which waiting thread is unblocked. If the specified message queue is full and `O_NONBLOCK` is set in the message queue description associated with `mqdes`, the message shall not be queued and `mq_send()` shall return an error.

The `mq_timedsend()` function shall add a message to the message queue specified by `mqdes` in the manner defined for the `mq_send()` function. However, if the specified message queue is full and `O_NONBLOCK` is not set in the message queue description associated with `mqdes`, the wait for sufficient room in the queue shall be terminated when the specified timeout expires. If `O_NONBLOCK` is set in the message queue description, this function shall be equivalent to `mq_send()`.

The timeout shall expire when the absolute time specified by `abstime` passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds `abstime`), or if the absolute time specified by `abstime` has already been passed at the time of the call.

The timeout shall be based on the `CLOCK_REALTIME` clock. The resolution of the timeout shall be the resolution of the clock on which it is based. The `timespec` argument is defined in the `<time.h>` header.

Under no circumstance shall the operation fail with a timeout if there is sufficient room in the queue to add the message immediately. The validity of the `abstime` parameter need not be checked when there is sufficient room in the queue.

mq_send()*System Interfaces***RETURN VALUE**

Upon successful completion, the *mq_send()* and *mq_timedsend()* functions shall return a value of zero. Otherwise, no message shall be enqueued, the functions shall return *-1*, and *errno* shall be set to indicate the error.

ERRORS

The *mq_send()* and *mq_timedsend()* functions shall fail if:

- [EAGAIN] The *O_NONBLOCK* flag is set in the message queue description associated with *mqdes*, and the specified message queue is full.
- [EBADF] The *mqdes* argument is not a valid message queue descriptor open for writing.
- [EINTR] A signal interrupted the call to *mq_send()* or *mq_timedsend()*.
- [EINVAL] The value of *msg_prio* was outside the valid range.
- [EINVAL] The process or thread would have blocked, and the *abstime* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.
- [EMSGSIZE] The specified message length, *msg_len*, exceeds the message size attribute of the message queue.
- [ETIMEDOUT] The *O_NONBLOCK* flag was not set when the message queue was opened, but the timeout expired before the message could be added to the queue.

EXAMPLES

None.

APPLICATION USAGE

The value of the symbol *{MQ_PRIO_MAX}* limits the number of priority levels supported by the application. Message priorities range from 0 to *{MQ_PRIO_MAX}-1*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open(), *mq_receive()*, *mq_setattr()*, *time()*

XBD *<mqqueue.h>*, *<time.h>*

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_send()* function is marked as part of the Message Passing option.

The *[ENOSYS]* error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The *mq_timedsend()* function is added for alignment with IEEE Std 1003.1d-1999.

Issue 7

The *mq_timedsend()* function is moved from the Timeouts option to the Base.

Functionality relating to the Timers option is moved to the Base.

System Interfaces

mq_setattr()**NAME**mq_setattr — set message queue attributes (**REALTIME**)**SYNOPSIS**

```
#include <mqueue.h>

int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
               struct mq_attr *restrict omqstat);
```

DESCRIPTION

The `mq_setattr()` function shall set attributes associated with the open message queue description referenced by the message queue descriptor specified by `mqdes`.

The message queue attributes corresponding to the following members defined in the **mq_attr** structure shall be set to the specified values upon successful completion of `mq_setattr()`:

mq_flags The value of this member is the bitwise-logical OR of zero or more of **O_NONBLOCK** and any implementation-defined flags.

The values of the `mq_maxmsg`, `mq_msgsize`, and `mq_curmsgs` members of the **mq_attr** structure shall be ignored by `mq_setattr()`.

If `omqstat` is non-NULL, the `mq_setattr()` function shall store, in the location referenced by `omqstat`, the previous message queue attributes and the current queue status. These values shall be the same as would be returned by a call to `mq_getattr()` at that point.

RETURN VALUE

Upon successful completion, the function shall return a value of zero and the attributes of the message queue shall have been changed as specified.

Otherwise, the message queue attributes shall be unchanged, and the function shall return a value of -1 and set `errno` to indicate the error.

ERRORS

The `mq_setattr()` function shall fail if:

[EBADF] The `mqdes` argument is not a valid message queue descriptor.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`mq_open()`, `mq_send()`, `msgctl()`, `msgget()`, `msgrcv()`, `msgsnd()`

XBD **<mqueue.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

mq_setattr()*System Interfaces*43967 **Issue 6**43968 The *mq_setattr()* function is marked as part of the Message Passing option.43969 The [ENOSYS] error condition has been removed as stubs need not be provided if an
43970 implementation does not support the Message Passing option.43971 The *mq_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std
43972 1003.1d-1999.43973 The **restrict** keyword is added to the *mq_setattr()* prototype for alignment with the
43974 ISO/IEC 9899:1999 standard.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***mq_timedreceive()**43975 **NAME**43976 `mq_timedreceive` — receive a message from a message queue (**ADVANCED REALTIME**)43977 **SYNOPSIS**

```
43978 MSG #include <mqueue.h>
43979 #include <time.h>
43980 ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
43981 size_t msg_len, unsigned *restrict msg_prio,
43982 const struct timespec *restrict abstime);
```

43983 **DESCRIPTION**43984 Refer to *mq_receive()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

mq_timedsend()*System Interfaces*43985 **NAME**43986 mq_timedsend — send a message to a message queue (**ADVANCED REALTIME**)43987 **SYNOPSIS**

```
43988 MSG      #include <mqueue.h>
43989           #include <time.h>
43990           int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
43991                           unsigned msg_prio, const struct timespec *abstime);
```

43992 **DESCRIPTION**43993 Refer to *mq_send()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

43994 NAME

43995 `mq_unlink` — remove a message queue (**REALTIME**)

43996 SYNOPSIS

```
43997 MSG #include <mqueue.h>
43998 int mq_unlink(const char *name);
```

43999 DESCRIPTION

44000 The `mq_unlink()` function shall remove the message queue named by the string `name`. If one or
 44001 more processes have the message queue open when `mq_unlink()` is called, destruction of the
 44002 message queue shall be postponed until all references to the message queue have been closed.
 44003 However, the `mq_unlink()` call need not block until all references have been closed; it may return
 44004 immediately.

44005 After a successful call to `mq_unlink()`, reuse of the name shall subsequently cause `mq_open()` to
 44006 behave as if no message queue of this name exists (that is, `mq_open()` will fail if `O_CREAT` is not
 44007 set, or will create a new message queue if `O_CREAT` is set).

44008 RETURN VALUE

44009 Upon successful completion, the function shall return a value of zero. Otherwise, the named
 44010 message queue shall be unchanged by this function call, and the function shall return a value of
 44011 `-1` and set `errno` to indicate the error.

44012 ERRORS

44013 The `mq_unlink()` function shall fail if:

44014 `[EACCES]` Permission is denied to unlink the named message queue.

44015 `[ENOENT]` The named message queue does not exist.

44016 The `mq_unlink()` function may fail if:

44017 `[ENAMETOOLONG]`

44018 The length of the `name` argument exceeds `{_POSIX_PATH_MAX}` on systems
 44019 XSI that do not support the XSI option or exceeds `{_XOPEN_PATH_MAX}` on XSI
 44020 systems, or has a pathname component that is longer than
 44021 XSI `{_POSIX_NAME_MAX}` on systems that do not support the XSI option or
 44022 longer than `{_XOPEN_NAME_MAX}` on XSI systems. A call to `mq_unlink()`
 44023 with a `name` argument that contains the same message queue name as was
 44024 previously used in a successful `mq_open()` call shall not give an
 44025 `[ENAMETOOLONG]` error.

44026 EXAMPLES

44027 None.

44028 APPLICATION USAGE

44029 None.

44030 RATIONALE

44031 None.

44032 FUTURE DIRECTIONS

44033 A future version might require the `mq_open()` and `mq_unlink()` functions to have semantics
 44034 similar to normal file system operations.

mq_unlink()

System Interfaces

44035 **SEE ALSO**44036 *mq_close()*, *mq_open()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*44037 XBD [<mqqueue.h>](#)44038 **CHANGE HISTORY**

44039 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

44040 **Issue 6**44041 The *mq_unlink()* function is marked as part of the Message Passing option.44042 The Open Group Corrigendum U021/5 is applied, clarifying that upon unsuccessful completion,
44043 the named message queue is unchanged by this function.44044 The [ENOSYS] error condition has been removed as stubs need not be provided if an
44045 implementation does not support the Message Passing option.44046 **Issue 7**44047 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a
44048 “shall fail” to a “may fail” error .

44049 Austin Group Interpretation 1003.1-2001 #141 is applied.

*System Interfaces***mrnd48()**44050 **NAME**

44051 mrnd48 — generate uniformly distributed pseudo-random signed long integers

44052 **SYNOPSIS**

```
44053 XSI      #include <stdlib.h>  
44054          long mrnd48(void);
```

44055 **DESCRIPTION**44056 Refer to *drand48()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

msgctl()*System Interfaces*44057 **NAME**

44058 msgctl — XSI message control operations

44059 **SYNOPSIS**

```
44060 XSI      #include <sys/msg.h>
44061      int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

44062 **DESCRIPTION**

44063 The *msgctl()* function operates on XSI message queues (see XBD [Section 3.224](#), on page 69). It is
 44064 unspecified whether this function interoperates with the realtime interprocess communication
 44065 facilities defined in [Section 2.8](#) (on page 497).

44066 The *msgctl()* function shall provide message control operations as specified by *cmd*. The
 44067 following values for *cmd*, and the message control operations they specify, are:

44068 IPC_STAT Place the current value of each member of the **msqid_ds** data structure
 44069 associated with *msqid* into the structure pointed to by *buf*. The contents of this
 44070 structure are defined in **<sys/msg.h>**.

44071 IPC_SET Set the value of the following members of the **msqid_ds** data structure
 44072 associated with *msqid* to the corresponding value found in the structure
 44073 pointed to by *buf*:

```
44074                   msg_perm.uid
44075                   msg_perm.gid
44076                   msg_perm.mode
44077                   msg_qbytes
```

44078 IPC_SET can only be executed by a process with appropriate privileges or that
 44079 has an effective user ID equal to the value of **msg_perm.cuid** or
 44080 **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*. Only a
 44081 process with appropriate privileges can raise the value of **msg_qbytes**.

44082 IPC_RMID Remove the message queue identifier specified by *msqid* from the system and
 44083 destroy the message queue and **msqid_ds** data structure associated with it.
 44084 IPC_RMID can only be executed by a process with appropriate privileges or
 44085 one that has an effective user ID equal to the value of **msg_perm.cuid** or
 44086 **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*.

44087 **RETURN VALUE**

44088 Upon successful completion, *msgctl()* shall return 0; otherwise, it shall return –1 and set *errno* to
 44089 indicate the error.

44090 **ERRORS**

44091 The *msgctl()* function shall fail if:

44092 [EACCES] The argument *cmd* is IPC_STAT and the calling process does not have read
 44093 permission; see [Section 2.7](#) (on page 496).

44094 [EINVAL] The value of *msqid* is not a valid message queue identifier; or the value of *cmd*
 44095 is not a valid command.

44096 [EPERM] The argument *cmd* is IPC_RMID or IPC_SET and the effective user ID of the
 44097 calling process is not equal to that of a process with appropriate privileges and
 44098 it is not equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the data
 44099 structure associated with *msqid*.

44100 [EPERM] The argument *cmd* is IPC_SET, an attempt is being made to increase to the
 44101 value of **msg_qbytes**, and the effective user ID of the calling process does not
 44102 have appropriate privileges.

44103 EXAMPLES

44104 None.

44105 APPLICATION USAGE

44106 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
 44107 (IPC). Application developers who need to use IPC should design their applications so that
 44108 modules using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to
 44109 use the alternative interfaces.

44110 RATIONALE

44111 None.

44112 FUTURE DIRECTIONS

44113 None.

44114 SEE ALSO

44115 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#),
 44116 [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#), [mq_unlink\(\)](#), [msgget\(\)](#), [msgrcv\(\)](#), [msgsnd\(\)](#)

44117 XBD [Section 3.224](#) (on page 69), [<sys/msg.h>](#)

44118 CHANGE HISTORY

44119 First released in Issue 2. Derived from Issue 2 of the SVID.

44120 Issue 5

44121 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
 44122 DIRECTIONS to a new APPLICATION USAGE section.

msgget()*System Interfaces*44123 **NAME**44124 **msgget** — get the XSI message queue identifier44125 **SYNOPSIS**

```
44126 XSI #include <sys/msg.h>
44127 int msgget(key_t key, int msgflg);
```

44128 **DESCRIPTION**

44129 The *msgget()* function operates on XSI message queues (see XBD [Section 3.224](#), on page 69). It is
 44130 unspecified whether this function interoperates with the realtime interprocess communication
 44131 facilities defined in [Section 2.8](#) (on page 497).

44132 The *msgget()* function shall return the message queue identifier associated with the argument
 44133 *key*.

44134 A message queue identifier, associated message queue, and data structure (see **<sys/msg.h>**),
 44135 shall be created for the argument *key* if one of the following is true:

- 44136 • The argument *key* is equal to `IPC_PRIVATE`.
- 44137 • The argument *key* does not already have a message queue identifier associated with it, and
 44138 (*msgflg* & `IPC_CREAT`) is non-zero.

44139 Upon creation, the data structure associated with the new message queue identifier shall be
 44140 initialized as follows:

- 44141 • **msg_perm.cuid**, **msg_perm.uid**, **msg_perm.cgid**, and **msg_perm.gid** shall be set equal to
 44142 the effective user ID and effective group ID, respectively, of the calling process.
- 44143 • The low-order 9 bits of **msg_perm.mode** shall be set equal to the low-order 9 bits of *msgflg*.
- 44144 • **msg_qnum**, **msg_lspid**, **msg_lrpid**, **msg_stime**, and **msg_rtime** shall be set equal to 0.
- 44145 • **msg_ctime** shall be set equal to the current time.
- 44146 • **msg_qbytes** shall be set equal to the system limit.

44147 **RETURN VALUE**

44148 Upon successful completion, *msgget()* shall return a non-negative integer, namely a message
 44149 queue identifier. Otherwise, it shall return `-1` and set *errno* to indicate the error.

44150 **ERRORS**

44151 The *msgget()* function shall fail if:

44152	[EACCES]	A message queue identifier exists for the argument <i>key</i> , but operation permission as specified by the low-order 9 bits of <i>msgflg</i> would not be granted; see Section 2.7 (on page 496).
44153		
44154		
44155	[EEXIST]	A message queue identifier exists for the argument <i>key</i> but ((<i>msgflg</i> & <code>IPC_CREAT</code>) && (<i>msgflg</i> & <code>IPC_EXCL</code>)) is non-zero.
44156		
44157	[ENOENT]	A message queue identifier does not exist for the argument <i>key</i> and (<i>msgflg</i> & <code>IPC_CREAT</code>) is 0.
44158		
44159	[ENOSPC]	A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system-wide would be exceeded.
44160		
44161		

44162 **EXAMPLES**

44163 None.

44164 **APPLICATION USAGE**

44165 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
 44166 (IPC). Application developers who need to use IPC should design their applications so that
 44167 modules using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to
 44168 use the alternative interfaces.

44169 **RATIONALE**

44170 None.

44171 **FUTURE DIRECTIONS**

44172 None.

44173 **SEE ALSO**

44174 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#),
 44175 [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#), [mq_unlink\(\)](#), [msgctl\(\)](#), [msgrcv\(\)](#), [msgsnd\(\)](#)

44176 XBD [Section 3.224](#) (on page 69), [<sys/msg.h>](#)44177 **CHANGE HISTORY**

44178 First released in Issue 2. Derived from Issue 2 of the SVID.

44179 **Issue 5**

44180 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
 44181 DIRECTIONS to a new APPLICATION USAGE section.

msgrcv()*System Interfaces*44182 **NAME**44183 **msgrcv** — XSI message receive operation44184 **SYNOPSIS**

```

44185 XSI      #include <sys/msg.h>
44186
44186 ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
44187               int msgflg);

```

44188 **DESCRIPTION**

44189 The *msgrcv()* function operates on XSI message queues (see XBD [Section 3.224](#), on page 69). It is
 44190 unspecified whether this function interoperates with the realtime interprocess communication
 44191 facilities defined in [Section 2.8](#) (on page 497).

44192 The *msgrcv()* function shall read a message from the queue associated with the message queue
 44193 identifier specified by *msqid* and place it in the user-defined buffer pointed to by *msgp*.

44194 The application shall ensure that the argument *msgp* points to a user-defined buffer that contains
 44195 first a field of type **long** specifying the type of the message, and then a data portion that holds
 44196 the data bytes of the message. The structure below is an example of what this user-defined
 44197 buffer might look like:

```

44198 struct mymsg {
44199     long    mtype;      /* Message type. */
44200     char    mtext[1];   /* Message text. */
44201 }

```

44202 The structure member *mtype* is the received message's type as specified by the sending process.

44203 The structure member *mtext* is the text of the message.

44204 The argument *msgsz* specifies the size in bytes of *mtext*. The received message shall be truncated
 44205 to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & MSG_NOERROR) is non-zero. The
 44206 truncated part of the message shall be lost and no indication of the truncation shall be given to
 44207 the calling process.

44208 If the value of *msgsz* is greater than {SSIZE_MAX}, the result is implementation-defined.

44209 The argument *msgtyp* specifies the type of message requested as follows:

- 44210 • If *msgtyp* is 0, the first message on the queue shall be received.
- 44211 • If *msgtyp* is greater than 0, the first message of type *msgtyp* shall be received.
- 44212 • If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the
 44213 absolute value of *msgtyp* shall be received.

44214 The argument *msgflg* specifies the action to be taken if a message of the desired type is not on the
 44215 queue. These are as follows:

- 44216 • If (*msgflg* & IPC_NOWAIT) is non-zero, the calling thread shall return immediately with a
 44217 return value of -1 and *errno* set to [ENOMSG].
- 44218 • If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the
 44219 following occurs:
 - 44220 — A message of the desired type is placed on the queue.
 - 44221 — The message queue identifier *msqid* is removed from the system; when this occurs,
 44222 *errno* shall be set equal to [EIDRM] and -1 shall be returned.

- The calling thread receives a signal that is to be caught; in this case a message is not received and the calling thread resumes execution in the manner prescribed in *sigaction()*.

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid*:

- **msg_qnum** shall be decremented by 1.
- **msg_lrpid** shall be set equal to the process ID of the calling process.
- **msg_rtime** shall be set equal to the current time.

RETURN VALUE

Upon successful completion, *msgrcv()* shall return a value equal to the number of bytes actually placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv()* shall return (**ssize_t**)-1, and *errno* shall be set to indicate the error.

ERRORS

The *msgrcv()* function shall fail if:

[E2BIG]	The value of <i>mtext</i> is greater than <i>msgsz</i> and (<i>msgflg</i> & MSG_NOERROR) is 0.
[EACCES]	Operation permission is denied to the calling process; see Section 2.7 (on page 496).
[EIDRM]	The message queue identifier <i>msqid</i> is removed from the system.
[EINTR]	The <i>msgrcv()</i> function was interrupted by a signal.
[EINVAL]	<i>msqid</i> is not a valid message queue identifier.
[ENOMSG]	The queue does not contain a message of the desired type and (<i>msgflg</i> & IPC_NOWAIT) is non-zero.

EXAMPLES

Receiving a Message

The following example receives the first message on the queue (based on the value of the *msgtyp* argument, 0). The queue is identified by the *msqid* argument (assuming that the value has previously been set). This call specifies that an error should be reported if no message is available, but not if the message is too large. The message size is calculated directly using the *sizeof* operator.

```
#include <sys/msg.h>
...
int result;
int msqid;
struct message {
    long type;
    char text[20];
} msg;
long msgtyp = 0;
...
result = msgrcv(msqid, (void *) &msg, sizeof(msg.text),
    msgtyp, MSG_NOERROR | IPC_NOWAIT);
```

msgrcv()*System Interfaces*44264 **APPLICATION USAGE**

44265 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
 44266 (IPC). Application developers who need to use IPC should design their applications so that
 44267 modules using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to
 44268 use the alternative interfaces.

44269 **RATIONALE**

44270 None.

44271 **FUTURE DIRECTIONS**

44272 None.

44273 **SEE ALSO**

44274 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#),
 44275 [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#), [mq_unlink\(\)](#), [msgctl\(\)](#), [msgget\(\)](#), [msgsnd\(\)](#),
 44276 [sigaction\(\)](#)

44277 XBD [Section 3.224](#) (on page 69), [<sys/msg.h>](#)

44278 **CHANGE HISTORY**

44279 First released in Issue 2. Derived from Issue 2 of the SVID.

44280 **Issue 5**

44281 The type of the return value is changed from **int** to **ssize_t**, and a warning is added to the
 44282 DESCRIPTION about values of *msgsz* larger the {SSIZE_MAX}.

44283 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
 44284 DIRECTIONS to the APPLICATION USAGE section.

44285 **Issue 6**

44286 The normative text is updated to avoid use of the term “must” for application requirements.

NAME

msgsnd — XSI message send operation

SYNOPSIS

```

#include <sys/msg.h>

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);

```

DESCRIPTION

The *msgsnd()* function operates on XSI message queues (see XBD [Section 3.224](#), on page 69). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in [Section 2.8](#) (on page 497).

The *msgsnd()* function shall send a message to the queue associated with the message queue identifier specified by *msqid*.

The application shall ensure that the argument *msgp* points to a user-defined buffer that contains first a field of type **long** specifying the type of the message, and then a data portion that holds the data bytes of the message. The structure below is an example of what this user-defined buffer might look like:

```

struct mymsg {
    long    mtype;        /* Message type. */
    char    mtext[1];     /* Message text. */
}

```

The structure member *mtype* is a non-zero positive type **long** that can be used by the receiving process for message selection.

The structure member *mtext* is any text of length *msgsz* bytes. The argument *msgsz* can range from 0 to a system-imposed maximum.

The argument *msgflg* specifies the action to be taken if one or more of the following is true:

- The number of bytes already on the queue is equal to **msg_qbytes**; see [<sys/msg.h>](#).
- The total number of messages on all queues system-wide is equal to the system-imposed limit.

These actions are as follows:

- If (*msgflg* & IPC_NOWAIT) is non-zero, the message shall not be sent and the calling thread shall return immediately.
- If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the following occurs:
 - The condition responsible for the suspension no longer exists, in which case the message is sent.
 - The message queue identifier *msqid* is removed from the system; when this occurs, *errno* shall be set equal to [EIDRM] and **-1** shall be returned.
 - The calling thread receives a signal that is to be caught; in this case the message is not sent and the calling thread resumes execution in the manner prescribed in [sigaction\(\)](#).

msgsnd()*System Interfaces*

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid*; see `<sys/msg.h>`:

- **msg_qnum** shall be incremented by 1.
- **msg_lspid** shall be set equal to the process ID of the calling process.
- **msg_stime** shall be set equal to the current time.

RETURN VALUE

Upon successful completion, *msgsnd()* shall return 0; otherwise, no message shall be sent, *msgsnd()* shall return -1, and *errno* shall be set to indicate the error.

ERRORS

The *msgsnd()* function shall fail if:

[EACCES]	Operation permission is denied to the calling process; see Section 2.7 (on page 496).
[EAGAIN]	The message cannot be sent for one of the reasons cited above and (<i>msgflg</i> & <i>IPC_NOWAIT</i>) is non-zero.
[EIDRM]	The message queue identifier <i>msqid</i> is removed from the system.
[EINTR]	The <i>msgsnd()</i> function was interrupted by a signal.
[EINVAL]	The value of <i>msqid</i> is not a valid message queue identifier, or the value of <i>mtype</i> is less than 1; or the value of <i>msgsz</i> is less than 0 or greater than the system-imposed limit.

EXAMPLES**Sending a Message**

The following example sends a message to the queue identified by the *msqid* argument (assuming that value has previously been set). This call specifies that an error should be reported if no message is available. The message size is calculated directly using the *sizeof* operator.

```
#include <sys/msg.h>
...
int result;
int msqid;
struct message {
    long type;
    char text[20];
} msg;

msg.type = 1;
strcpy(msg.text, "This is message 1");
...
result = msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);
```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to use the alternative interfaces.

44367 **RATIONALE**

44368 None.

44369 **FUTURE DIRECTIONS**

44370 None.

44371 **SEE ALSO**

44372 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#),
 44373 [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#), [mq_unlink\(\)](#), [msgctl\(\)](#), [msgget\(\)](#), [msgrcv\(\)](#),
 44374 [sigaction\(\)](#)

44375 XBD [Section 3.224](#) (on page 69), [<sys/msg.h>](#)44376 **CHANGE HISTORY**

44377 First released in Issue 2. Derived from Issue 2 of the SVID.

44378 **Issue 5**

44379 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
 44380 DIRECTIONS to a new APPLICATION USAGE section.

44381 **Issue 6**

44382 The normative text is updated to avoid use of the term “must” for application requirements.

msync()

NAME

msync — synchronize memory with physical storage

SYNOPSIS

```
#include <sys/mman.h>

int msync(void *addr, size_t len, int flags);
```

DESCRIPTION

The *msync()* function shall write all modified data to permanent storage locations, if any, in those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes. If no such storage exists, *msync()* need not have any effect. If requested, the *msync()* function shall then invalidate cached copies of data.

The implementation may require that *addr* be a multiple of the page size as returned by *sysconf()*.

For mappings to files, the *msync()* function shall ensure that all write operations are completed as defined for synchronized I/O data integrity completion. It is unspecified whether the implementation also writes out other file attributes. When the *msync()* function is called on MAP_PRIVATE mappings, any modified data shall not be written to the underlying object and shall not cause such data to be made visible to other processes. It is unspecified whether data in MAP_PRIVATE mappings has any permanent storage locations. The effect of *msync()* on a shared memory object or a typed memory object is unspecified. The behavior of this function is unspecified if the mapping was not established by a call to *mmap()*.

The *flags* argument is constructed from the bitwise-inclusive OR of one or more of the following flags defined in the *<sys/mman.h>* header:

Symbolic Constant	Description
MS_ASYNC	Perform asynchronous writes.
MS_SYNC	Perform synchronous writes.
MS_INVALIDATE	Invalidate cached data.

When MS_ASYNC is specified, *msync()* shall return immediately once all the write operations are initiated or queued for servicing; when MS_SYNC is specified, *msync()* shall not return until all write operations are completed as defined for synchronized I/O data integrity completion. Either MS_ASYNC or MS_SYNC shall be specified, but not both.

When MS_INVALIDATE is specified, *msync()* shall invalidate all cached copies of mapped data that are inconsistent with the permanent storage locations such that subsequent references shall obtain data that was consistent with the permanent storage locations sometime between the call to *msync()* and the first subsequent memory reference to the data.

If *msync()* causes any write to a file, the file's last data modification and last file status change timestamps shall be marked for update.

RETURN VALUE

Upon successful completion, *msync()* shall return 0; otherwise, it shall return -1 and set *errno* to indicate the error.

ERRORS

The *msync()* function shall fail if:

[EBUSY] Some or all of the addresses in the range starting at *addr* and continuing for *len* bytes are locked, and MS_INVALIDATE is specified.

- 44426 [EINVAL] The value of *flags* is invalid.
- 44427 [ENOMEM] The addresses in the range starting at *addr* and continuing for *len* bytes are
- 44428 outside the range allowed for the address space of a process or specify one or
- 44429 more pages that are not mapped.

44430 The *msync()* function may fail if:

- 44431 [EINVAL] The value of *addr* is not a multiple of the page size as returned by *sysconf()*.

44432 EXAMPLES

44433 None.

44434 APPLICATION USAGE

44435 The *msync()* function is only supported if the Synchronized Input and Output option is

44436 supported, and thus need not be available on all implementations.

44437 The *msync()* function should be used by programs that require a memory object to be in a

44438 known state; for example, in building transaction facilities.

44439 Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees

44440 that *msync()* is the only control over when pages are or are not written to disk.

44441 RATIONALE

44442 The *msync()* function writes out data in a mapped region to the permanent storage for the

44443 underlying object. The call to *msync()* ensures data integrity of the file.

44444 After the data is written out, any cached data may be invalidated if the MS_INVALIDATE flag

44445 was specified. This is useful on systems that do not support read/write consistency.

44446 FUTURE DIRECTIONS

44447 None.

44448 SEE ALSO

44449 *mmap()*, *sysconf()*

44450 XBD <sys/mman.h>

44451 CHANGE HISTORY

44452 First released in Issue 4, Version 2.

44453 Issue 5

44454 Moved from X/OPEN UNIX extension to BASE.

44455 Aligned with *msync()* in the POSIX Realtime Extension as follows:

- 44456 • The DESCRIPTION is extensively reworded.
- 44457 • [EBUSY] and a new form of [EINVAL] are added as mandatory error conditions.

44458 Issue 6

44459 The *msync()* function is marked as part of the Memory Mapped Files and Synchronized Input

44460 and Output options.

44461 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 44462 • The [EBUSY] mandatory error condition is added.

44463 The following new requirements on POSIX implementations derive from alignment with the

44464 Single UNIX Specification:

msync()*System Interfaces*

- 44465 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple
44466 of the page size.
- 44467 • The second [EINVAL] error condition is made mandatory.
- 44468 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding reference to
44469 typed memory objects.
- 44470 **Issue 7**
- 44471 Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment
44472 requirements.
- 44473 SD5-XSH-ERN-110 is applied.
- 44474 The *msync()* function is marked as part of the Synchronized Input and Output option or XSI
44475 option as the Memory Mapped Files is moved to the Base.
- 44476 Changes are made related to support for finegrained timestamps.

*System Interfaces***munlock()**44477 **NAME**

44478 munlock — unlock a range of process address space

44479 **SYNOPSIS**

44480 MLR #include <sys/mman.h>

44481 int munlock(const void *addr, size_t len);

44482 **DESCRIPTION**44483 Refer to *mlock()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

munlockall()*System Interfaces*44484 **NAME**

44485 munlockall — unlock the address space of a process

44486 **SYNOPSIS**

```
44487 ML     #include <sys/mman.h>
44488     int munlockall(void);
```

44489 **DESCRIPTION**44490 Refer to *mlinkall()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

munmap()**NAME**

munmap — unmap pages of memory

SYNOPSIS

```
#include <sys/mman.h>
```

```
int munmap(void *addr, size_t len);
```

DESCRIPTION

The *munmap()* function shall remove any mappings for those entire pages containing any part of the address space of the process starting at *addr* and continuing for *len* bytes. Further references to these pages shall result in the generation of a SIGSEGV signal to the process. If there are no mappings in the specified address range, then *munmap()* has no effect.

The implementation may require that *addr* be a multiple of the page size as returned by *sysconf()*.

If a mapping to be removed was private, any modifications made in this address range shall be discarded.

ML|MLR Any memory locks (see *mlock()* and *mlockall()*) associated with this address range shall be removed, as if by an appropriate call to *munlock()*.

TYM If a mapping removed from a typed memory object causes the corresponding address range of the memory pool to be inaccessible by any process in the system except through allocatable mappings (that is, mappings of typed memory objects opened with the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag), then that range of the memory pool shall become deallocated and may become available to satisfy future typed memory allocation requests.

A mapping removed from a typed memory object opened with the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag shall not affect in any way the availability of that typed memory for allocation.

The behavior of this function is unspecified if the mapping was not established by a call to *mmap()*.

RETURN VALUE

Upon successful completion, *munmap()* shall return 0; otherwise, it shall return -1 and set *errno* to indicate the error.

ERRORS

The *munmap()* function shall fail if:

[EINVAL] Addresses in the range [*addr*,*addr*+*len*) are outside the valid range for the address space of a process.

[EINVAL] The *len* argument is 0.

The *munmap()* function may fail if:

[EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf()*.

munmap()*System Interfaces*44528 **EXAMPLES**

44529 None.

44530 **APPLICATION USAGE**

44531 None.

44532 **RATIONALE**44533 The *munmap()* function corresponds to SVR4, just as the *mmap()* function does.

44534 It is possible that an application has applied process memory locking to a region that contains
 44535 shared memory. If this has occurred, the *munmap()* call ignores those locks and, if necessary,
 44536 causes those locks to be removed.

44537 Most implementations require that *addr* is a multiple of the page size as returned by *sysconf()*.44538 **FUTURE DIRECTIONS**

44539 None.

44540 **SEE ALSO**44541 *mlock()*, *mlockall()*, *mmap()*, *posix_typed_mem_open()*, *sysconf()*

44542 XBD <sys/mman.h>

44543 **CHANGE HISTORY**

44544 First released in Issue 4, Version 2.

44545 **Issue 5**

44546 Moved from X/OPEN UNIX extension to BASE.

44547 Aligned with *munmap()* in the POSIX Realtime Extension as follows:

- 44548 • The DESCRIPTION is extensively reworded.
- 44549 • The SIGBUS error is no longer permitted to be generated.

44550 **Issue 6**

44551 The *munmap()* function is marked as part of the Memory Mapped Files and Shared Memory
 44552 Objects option.

44553 The following new requirements on POSIX implementations derive from alignment with the
 44554 Single UNIX Specification:

- 44555 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple
 44556 of the page size.
- 44557 • The [EINVAL] error conditions are added.

44558 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 44559 • Semantics for typed memory objects are added to the DESCRIPTION.
- 44560 • The *posix_typed_mem_open()* function is added to the SEE ALSO section.

44561 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code
 44562 in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM).

44563 **Issue 7**

44564 Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment
 44565 requirements.

44566 The *munmap()* function is moved from the Memory Mapped Files option to the Base.

NAME

nan, nanf, nanl — return quiet NaN

SYNOPSIS

```
#include <math.h>

double nan(const char *tagp);
float nanf(const char *tagp);
long double nanl(const char *tagp);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The function call *nan*("n-char-sequence") shall be equivalent to:

```
strtod("NAN(n-char-sequence)", (char **) NULL);
```

The function call *nan*("") shall be equivalent to:

```
strtod("NAN()", (char **) NULL)
```

If *tagp* does not point to an *n-char* sequence or an empty string, the function call shall be equivalent to:

```
strtod("NAN", (char **) NULL)
```

Function calls to *nanf*() and *nanl*() are equivalent to the corresponding function calls to *strtof*() and *strtold*().

RETURN VALUE

These functions shall return a quiet NaN, if available, with content indicated through *tagp*.

If the implementation does not support quiet NaNs, these functions shall return zero.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtod()

XBD *<math.h>*

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

nanosleep()

System Interfaces

44605 **NAME**

44606 nanosleep — high resolution sleep

44607 **SYNOPSIS**

```
44608 CX #include <time.h>
44609 int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
```

44610 **DESCRIPTION**

44611 The *nanosleep()* function shall cause the current thread to be suspended from execution until
 44612 either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the
 44613 calling thread, and its action is to invoke a signal-catching function or to terminate the process.
 44614 The suspension time may be longer than requested because the argument value is rounded up to
 44615 an integer multiple of the sleep resolution or because of the scheduling of other activity by the
 44616 system. But, except for the case of being interrupted by a signal, the suspension time shall not be
 44617 less than the time specified by *rqtp*, as measured by the system clock *CLOCK_REALTIME*.

44618 The use of the *nanosleep()* function has no effect on the action or blockage of any signal.

44619 **RETURN VALUE**

44620 If the *nanosleep()* function returns because the requested time has elapsed, its return value shall
 44621 be zero.

44622 If the *nanosleep()* function returns because it has been interrupted by a signal, it shall return a
 44623 value of *-1* and set *errno* to indicate the interruption. If the *rmtp* argument is non-NULL, the
 44624 **timespec** structure referenced by it is updated to contain the amount of time remaining in the
 44625 interval (the requested time minus the time actually slept). The *rqtp* and *rmtp* arguments may
 44626 point to the same object. If the *rmtp* argument is NULL, the remaining time is not returned.

44627 If *nanosleep()* fails, it shall return a value of *-1* and set *errno* to indicate the error.

44628 **ERRORS**

44629 The *nanosleep()* function shall fail if:

44630	[EINTR]	The <i>nanosleep()</i> function was interrupted by a signal.
44631	[EINVAL]	The <i>rqtp</i> argument specified a nanosecond value less than zero or greater than or equal to 1 000 million.
44632		

44633 **EXAMPLES**

44634 None.

44635 **APPLICATION USAGE**

44636 None.

44637 **RATIONALE**

44638 It is common to suspend execution of a thread for an interval in order to poll the status of a non-
 44639 interrupting function. A large number of actual needs can be met with a simple extension to
 44640 *sleep()* that provides finer resolution.

44641 In the POSIX.1-1990 standard and SVR4, it is possible to implement such a routine, but the
 44642 frequency of wakeup is limited by the resolution of the *alarm()* and *sleep()* functions. In 4.3 BSD,
 44643 it is possible to write such a routine using no static storage and reserving no system facilities.
 44644 Although it is possible to write a function with similar functionality to *sleep()* using the
 44645 remainder of the *timer_**() functions, such a function requires the use of signals and the
 44646 reservation of some signal number. This volume of POSIX.1-2008 requires that *nanosleep()* be
 44647 non-intrusive of the signals function.

44648 The *nanosleep()* function shall return a value of 0 on success and *-1* on failure or if interrupted.

44649 This latter case is different from *sleep()*. This was done because the remaining time is returned
 44650 via an argument structure pointer, *rmtp*, instead of as the return value.

44651 FUTURE DIRECTIONS

44652 None.

44653 SEE ALSO

44654 *clock_nanosleep()*, *sleep()*

44655 XBD <time.h>

44656 CHANGE HISTORY

44657 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

44658 Issue 6

44659 The *nanosleep()* function is marked as part of the Timers option.

44660 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 44661 implementation does not support the Timers option.

44662 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/37 is applied, updating the SEE ALSO
 44663 section to include the *clock_nanosleep()* function.

44664 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/63 is applied, correcting text in the
 44665 RATIONALE section.

44666 Issue 7

44667 SD5-XBD-ERN-33 is applied.

44668 The *nanosleep()* function is moved from the Timers option to the Base.

nearbyint()*System Interfaces*44669 **NAME**44670 `nearbyint`, `nearbyintf`, `nearbyintl` — floating-point rounding functions44671 **SYNOPSIS**

```
44672 #include <math.h>
44673 double nearbyint(double x);
44674 float nearbyintf(float x);
44675 long double nearbyintl(long double x);
```

44676 **DESCRIPTION**

44677 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 44678 conflict between the requirements described here and the ISO C standard is unintentional. This
 44679 volume of POSIX.1-2008 defers to the ISO C standard.

44680 These functions shall round their argument to an integer value in floating-point format, using
 44681 the current rounding direction and without raising the inexact floating-point exception.

44682 An application wishing to check for error situations should set *errno* to zero and call
 44683 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 44684 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 44685 zero, an error has occurred.

44686 **RETURN VALUE**

44687 Upon successful completion, these functions shall return the rounded integer value.

44688 MX If *x* is NaN, a NaN shall be returned.

44689 If *x* is ± 0 , ± 0 shall be returned.

44690 If *x* is $\pm \text{Inf}$, *x* shall be returned.

44691 XSI If the correct value would cause overflow, a range error shall occur and *nearbyint*(), *nearbyintf*(),
 44692 and *nearbyintl*() shall return the value of the macro $\pm \text{HUGE\_VAL}$, $\pm \text{HUGE\_VALF}$, and
 44693 $\pm \text{HUGE\_VALL}$ (with the same sign as *x*), respectively.

44694 **ERRORS**

44695 These functions shall fail if:

44696 XSI **Range Error** The result would cause an overflow.

44697 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 44698 then *errno* shall be set to [ERANGE]. If the integer expression
 44699 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 44700 floating-point exception shall be raised.

44701 **EXAMPLES**

44702 None.

44703 **APPLICATION USAGE**

44704 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 44705 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

44706 **RATIONALE**

44707 None.

44708 **FUTURE DIRECTIONS**

44709 None.

44710 **SEE ALSO**44711 [fclearexcept\(\)](#), [fetestexcept\(\)](#)44712 XBD [Section 4.19](#) (on page 116), [<math.h>](#)44713 **CHANGE HISTORY**

44714 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

newlocale()

System Interfaces

44715 **NAME**44716 **newlocale** — create or modify a locale object44717 **SYNOPSIS**

```

44718 CX    #include <locale.h>
44719
44719    locale_t newlocale(int category_mask, const char *locale,
44720                      locale_t base);

```

44721 **DESCRIPTION**

44722 The *newlocale()* function shall create a new locale object or modify an existing one. If the *base*
 44723 argument is (**locale_t**)0, a new locale object shall be created. It is unspecified whether the locale
 44724 object pointed to by *base* shall be modified or freed and a new locale object created.

44725 The *category_mask* argument specifies the locale categories to be set or modified. Values for
 44726 *category_mask* shall be constructed by a bitwise-inclusive OR of the symbolic constants
 44727 *LC_CTYPE_MASK*, *LC_NUMERIC_MASK*, *LC_TIME_MASK*, *LC_COLLATE_MASK*,
 44728 *LC_MONETARY_MASK*, and *LC_MESSAGES_MASK*, or any of the other implementation-
 44729 defined *LC_*_MASK* values defined in **<locale.h>**.

44730 For each category with the corresponding bit set in *category_mask* the data from the locale named
 44731 by *locale* shall be used. In the case of modifying an existing locale object, the data from the locale
 44732 named by *locale* shall replace the existing data within the locale object. If a completely new locale
 44733 object is created, the data for all sections not requested by *category_mask* shall be taken from the
 44734 default locale.

44735 The following preset values of *locale* are defined for all settings of *category_mask*:

44736	"POSIX"	Specifies the minimal environment for C-language translation called the
44737		POSIX locale.
44738	"C"	Equivalent to "POSIX".
44739	""	Specifies an implementation-defined native environment. This corresponds to
44740		the value of the associated environment variables, <i>LC_*</i> and <i>LANG</i> ; see XBD
44741		Chapter 7 (on page 135) and Chapter 8 (on page 173).

44742 If the *base* argument is not (**locale_t**)0 and the *newlocale()* function call succeeds, the contents of
 44743 *base* are unspecified. Applications shall ensure that they stop using *base* as a locale object before
 44744 calling *newlocale()*. If the function call fails and the *base* argument is not (**locale_t**)0, the contents
 44745 of *base* shall remain valid and unchanged.

44746 The results are undefined if the *base* argument is the special locale object *LC_GLOBAL_LOCALE*.

44747 **RETURN VALUE**

44748 Upon successful completion, the *newlocale()* function shall return a handle which the caller may
 44749 use on subsequent calls to *duplocale()*, *freelocale()*, and other functions taking a **locale_t**
 44750 argument.

44751 Upon failure, the *newlocale()* function shall return (**locale_t**)0 and set *errno* to indicate the error.

44752 **ERRORS**

44753 The *newlocale()* function shall fail if:

44754	[ENOMEM]	There is not enough memory available to create the locale object or load the
44755		locale data.

[EINVAL] The *category_mask* contains a bit that does not correspond to a valid category.

[ENOENT] For any of the categories in *category_mask*, the locale data is not available.

The *newlocale()* function may fail if:

[EINVAL] The *locale* argument is not a valid string pointer.

EXAMPLES

Constructing a Locale Object from Different Locales

The following example shows the construction of a locale where the *LC_CTYPE* category data comes from a locale *loc1* and the *LC_TIME* category data from a locale *tok2*:

```
#include <locale.h>
...
locale_t loc, new_loc;

/* Get the "loc1" data. */

loc = newlocale (LC_CTYPE_MASK, "loc1", NULL);
if (loc == (locale_t) 0)
    abort ();

/* Get the "loc2" data. */

new_loc = newlocale (LC_TIME_MASK, "loc2", loc);
if (new_loc != (locale_t) 0)
    /* We don't abort if this fails. In this case this
       simply used to unchanged locale object. */
    loc = new_loc;
...
```

Freeing up a Locale Object

The following example shows a code fragment to free a locale object created by *newlocale()*:

```
#include <locale.h>
...

/* Every locale object allocated with newlocale() should be
   * freed using freelocale():
   */

locale_t loc;

/* Get the locale. */

loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);

/* ... Use the locale object ... */
...

/* Free the locale object resources. */
freelocale (loc);
```

newlocale()*System Interfaces*44792 **APPLICATION USAGE**

44793 Handles for locale objects created by the *newlocale()* function should be released by a
44794 corresponding call to *freelocale()*.

44795 The special locale object *LC_GLOBAL_LOCALE* must not be passed for the *base* argument, even
44796 when returned by the *uselocale()* function.

44797 **RATIONALE**

44798 None.

44799 **FUTURE DIRECTIONS**

44800 None.

44801 **SEE ALSO**

44802 *duplocale()*, *freelocale()*, *uselocale()*

44803 XBD [Chapter 7](#) (on page 135), [Chapter 8](#) (on page 173), [<locale.h>](#)

44804 **CHANGE HISTORY**

44805 First released in Issue 7.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

nextafter, *nextafterf*, *nextafterl*, *nexttoward*, *nexttowardf*, *nexttowardl* — next representable floating-point number

SYNOPSIS

```
#include <math.h>

double nextafter(double x, double y);
float nextafterf(float x, float y);
long double nextafterl(long double x, long double y);
double nexttoward(double x, long double y);
float nexttowardf(float x, long double y);
long double nexttowardl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *nextafter*(), *nextafterf*(), and *nextafterl*() functions shall compute the next representable floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, *nextafter*() shall return the largest representable floating-point number less than *x*. The *nextafter*(), *nextafterf*(), and *nextafterl*() functions shall return *y* if *x* equals *y*.

The *nexttoward*(), *nexttowardf*(), and *nexttowardl*() functions shall be equivalent to the corresponding *nextafter*() functions, except that the second parameter shall have type **long double** and the functions shall return *y* converted to the type of the function if *x* equals *y*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the next representable floating-point value following *x* in the direction of *y*.

If *x*==*y*, *y* (of the type *x*) shall be returned.

If *x* is finite and the correct function value would overflow, a range error shall occur and ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (with the same sign as *x*) shall be returned as appropriate for the return type of the function.

MX If *x* or *y* is NaN, a NaN shall be returned.

If *x*!=*y* and the correct function value is subnormal, zero, or underflows, a range error shall occur, and either the correct function value (if representable) or 0.0 shall be returned.

ERRORS

These functions shall fail if:

Range Error The correct value overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

nextafter()*System Interfaces*

44849	MX	Range Error	The correct value is subnormal or underflows.
44850			If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
44851			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
44852			(<i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the underflow
44853			floating-point exception shall be raised.
44854		EXAMPLES	
44855		None.	
44856		APPLICATION USAGE	
44857		On error, the expressions (<i>math_errhandling</i> & MATH_ERRNO) and (<i>math_errhandling</i> &	
44858		MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.	
44859		RATIONALE	
44860		None.	
44861		FUTURE DIRECTIONS	
44862		None.	
44863		SEE ALSO	
44864		<i>feclearexcept()</i> , <i>fetestexcept()</i>	
44865		XBD Section 4.19 (on page 116), <math.h>	
44866		CHANGE HISTORY	
44867		First released in Issue 4, Version 2.	
44868		Issue 5	
44869		Moved from X/OPEN UNIX extension to BASE.	
44870		Issue 6	
44871		The <i>nextafter()</i> function is no longer marked as an extension.	
44872		The <i>nextafterf()</i> , <i>nextafterl()</i> , <i>nexttoward()</i> , <i>nexttowardf()</i> , and <i>nexttowardl()</i> functions are added	
44873		for alignment with the ISO/IEC 9899:1999 standard.	
44874		The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are	
44875		revised to align with the ISO/IEC 9899:1999 standard.	
44876		IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are	
44877		marked.	

System Interfaces

nftw()**NAME**

nftw — walk a file tree

SYNOPSIS

```

#include <ftw.h>

int nftw(const char *path, int (*fn)(const char *,
    const struct stat *, int, struct FTW *), int fd_limit, int flags);

```

DESCRIPTION

The *nftw()* function shall recursively descend the directory hierarchy rooted in *path*. The *nftw()* function has a similar effect to *ftw()* except that it takes an additional argument *flags*, which is a bitwise-inclusive OR of zero or more of the following flags:

FTW_CHDIR	If set, <i>nftw()</i> shall change the current working directory to each directory as it reports files in that directory. If clear, <i>nftw()</i> shall not change the current working directory.
FTW_DEPTH	If set, <i>nftw()</i> shall report all files in a directory before reporting the directory itself. If clear, <i>nftw()</i> shall report any directory before reporting the files in that directory.
FTW_MOUNT	If set, <i>nftw()</i> shall only report files in the same file system as <i>path</i> . If clear, <i>nftw()</i> shall report all files encountered during the walk.
FTW_PHYS	If set, <i>nftw()</i> shall perform a physical walk and shall not follow symbolic links.

If FTW_PHYS is clear and FTW_DEPTH is set, *nftw()* shall follow links instead of reporting them, but shall not report any directory that would be a descendant of itself. If FTW_PHYS is clear and FTW_DEPTH is clear, *nftw()* shall follow links instead of reporting them, but shall not report the contents of any directory that would be a descendant of itself.

At each file it encounters, *nftw()* shall call the user-supplied function *fn* with four arguments:

- The first argument is the pathname of the object.
- The second argument is a pointer to the **stat** buffer containing information on the object, filled in as if *fstat()*, *stat()*, or *lstat()* had been called to retrieve the information.
- The third argument is an integer giving additional information. Its value is one of the following:

FTW_D	The object is a directory.
FTW_DNR	The object is a directory that cannot be read. The <i>fn</i> function shall not be called for any of its descendants.
FTW_DP	The object is a directory and subdirectories have been visited. (This condition shall only occur if the FTW_DEPTH flag is included in <i>flags</i> .)
FTW_F	The object is a file.
FTW_NS	The <i>stat()</i> function failed on the object because of lack of appropriate permission. The stat buffer passed to <i>fn</i> is undefined. Failure of <i>stat()</i> for any other reason is considered an error and <i>nftw()</i> shall return <i>-1</i> .
FTW_SL	The object is a symbolic link. (This condition shall only occur if the FTW_PHYS flag is included in <i>flags</i> .)

nftw()*System Interfaces*

44918 FTW_SLN The object is a symbolic link that does not name an existing file. (This
44919 condition shall only occur if the FTW_PHYS flag is not included in *flags*.)

- 44920 • The fourth argument is a pointer to an **FTW** structure. The value of **base** is the offset of the
44921 object's filename in the pathname passed as the first argument to *fn*. The value of **level**
44922 indicates depth relative to the root of the walk, where the root level is 0.

44923 The results are unspecified if the application-supplied *fn* function does not preserve the current
44924 working directory.

44925 The argument *fd_limit* sets the maximum number of file descriptors that shall be used by *nftw()*
44926 while traversing the file tree. At most one file descriptor shall be used for each directory level.

44927 The *nftw()* function need not be thread-safe.

RETURN VALUE

44928 The *nftw()* function shall continue until the first of the following conditions occurs:

- 44930 • An invocation of *fn* shall return a non-zero value, in which case *nftw()* shall return that
44931 value.
- 44932 • The *nftw()* function detects an error other than [EACCES] (see FTW_DNR and FTW_NS
44933 above), in which case *nftw()* shall return -1 and set *errno* to indicate the error.
- 44934 • The tree is exhausted, in which case *nftw()* shall return 0.

ERRORS

44935 The *nftw()* function shall fail if:

44937 [EACCES] Search permission is denied for any component of *path* or read permission is
44938 denied for *path*, or *fn* returns -1 and does not reset *errno*.

44939 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
44940 argument.

44941 [ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

44943 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

44944 [ENOTDIR] A component of *path* is not a directory.

44945 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current
44946 programming environment for one or more files found in the file hierarchy.

44947 The *nftw()* function may fail if:

44948 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
44949 resolution of the *path* argument.

44950 [EMFILE] All file descriptors available to the process are currently open.

44951 [ENAMETOOLONG] The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
44952 symbolic link produced an intermediate result with a length that exceeds
44953 {PATH_MAX}.
44954

44955 [ENFILE] Too many files are currently open in the system.

44956 In addition, *errno* may be set if the function pointed to by *fn* causes *errno* to be set.

EXAMPLES

The following program traverses the directory tree under the path named in its first command-line argument, or under the current directory if no argument is supplied. It displays various information about each file. The second command-line argument can be used to specify characters that control the value assigned to the *flags* argument when calling *nftw()*.

```
#include <ftw.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

static int
display_info(const char *fpath, const struct stat *sb,
             int tflag, struct FTW *ftwbuf)
{
    printf("%-3s %2d %7jd    %-40s %d %s\n",
           (tflag == FTW_D) ? "d" : (tflag == FTW_DNR) ? "dnr" :
           (tflag == FTW_DP) ? "dp" : (tflag == FTW_F) ? "f" :
           (tflag == FTW_NS) ? "ns" : (tflag == FTW_SL) ? "sl" :
           (tflag == FTW_SLN) ? "sln" : "???",
           ftwbuf->level, (intmax_t) sb->st_size,
           fpath, ftwbuf->base, fpath + ftwbuf->base);
    return 0; /* To tell nftw() to continue */
}

int
main(int argc, char *argv[])
{
    int flags = 0;

    if (argc > 2 && strchr(argv[2], 'd') != NULL)
        flags |= FTW_DEPTH;
    if (argc > 2 && strchr(argv[2], 'p') != NULL)
        flags |= FTW_PHYS;

    if (nftw((argc < 2) ? "." : argv[1], display_info, 20, flags) == -1)
    {
        perror("nftw");
        exit(EXIT_FAILURE);
    }

    exit(EXIT_SUCCESS);
}
```

APPLICATION USAGE

The *nftw()* function may allocate dynamic storage during its operation. If *nftw()* is forcibly terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *fn* or an interrupt routine, *nftw()* does not have a chance to free that storage, so it remains permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next invocation.

nftw()*System Interfaces*45002 **RATIONALE**

45003 None.

45004 **FUTURE DIRECTIONS**

45005 None.

45006 **SEE ALSO**45007 *fdopendir()*, *fstatat()*, *readdir()*

45008 XBD <ftw.h>

45009 **CHANGE HISTORY**

45010 First released in Issue 4, Version 2.

45011 **Issue 5**

45012 Moved from X/OPEN UNIX extension to BASE.

45013 In the DESCRIPTION, the definition of the *depth* argument is clarified.45014 **Issue 6**

45015 The Open Group Base Resolution bwg97-003 is applied.

45016 The ERRORS section is updated as follows:

- 45017 • The wording of the mandatory [ELOOP] error condition is updated.
- 45018 • A second optional [ELOOP] error condition is added.
- 45019 • The [Eoverflow] mandatory error condition is added.

45020 Text is added to the DESCRIPTION to say that the *nftw()* function need not be reentrant and that

45021 the results are unspecified if the application-supplied *fn* function does not preserve the current

45022 working directory.

45023 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/64 is applied, changing the argument

45024 *depth* to *fd_limit* throughout and changing “to a maximum of 5 levels deep” to “using a

45025 maximum of 5 file descriptors” in the EXAMPLES section.

45026 **Issue 7**

45027 Austin Group Interpretations 1003.1-2001 #143 and #156 are applied.

45028 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

45029 SD5-XBD-ERN-61 is applied.

45030 APPLICATION USAGE is added and the EXAMPLES section is replaced with a new example.

NAME

nice — change the nice value of a process

SYNOPSIS

```
#include <unistd.h>

int nice(int incr);
```

DESCRIPTION

The *nice()* function shall add the value of *incr* to the nice value of the calling process. A nice value of a process is a non-negative number for which a more positive value shall result in less favorable scheduling.

A maximum nice value of $2 \times \{\text{NZERO}\} - 1$ and a minimum nice value of 0 shall be imposed by the system. Requests for values above or below these limits shall result in the nice value being set to the corresponding limit. Only a process with appropriate privileges can lower the nice value.

Calling the *nice()* function has no effect on the priority of processes or threads with policy SCHED_FIFO or SCHED_RR. The effect on processes or threads with other scheduling policies is implementation-defined.

The nice value set with *nice()* shall be applied to the process. If the process is multi-threaded, the nice value shall affect all system scope threads in the process.

As -1 is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *nice()*, and if it returns -1, check to see whether *errno* is non-zero.

RETURN VALUE

Upon successful completion, *nice()* shall return the new nice value $-\{\text{NZERO}\}$. Otherwise, -1 shall be returned, the nice value of the process shall not be changed, and *errno* shall be set to indicate the error.

ERRORS

The *nice()* function shall fail if:

[EPERM] The *incr* argument is negative and the calling process does not have appropriate privileges.

EXAMPLES**Changing the Nice Value**

The following example adds the value of the *incr* argument, -20, to the nice value of the calling process.

```
#include <unistd.h>
...
int incr = -20;
int ret;

ret = nice(incr);
```

APPLICATION USAGE

None.

nice()*System Interfaces*45070 **RATIONALE**

45071 None.

45072 **FUTURE DIRECTIONS**

45073 None.

45074 **SEE ALSO**45075 *exec, getpriority()*45076 XBD *<limits.h>, <unistd.h>*45077 **CHANGE HISTORY**

45078 First released in Issue 1. Derived from Issue 1 of the SVID.

45079 **Issue 5**45080 A statement is added to the description indicating the effects of this function on the different
45081 scheduling policies and multi-threaded processes.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

nl_langinfo, nl_langinfo_l — language information

SYNOPSIS

```
#include <langinfo.h>

char *nl_langinfo(nl_item item);
char *nl_langinfo_l(nl_item item, locale_t locale);
```

DESCRIPTION

The *nl_langinfo()* and *nl_langinfo_l()* functions shall return a pointer to a string containing information relevant to the particular language or cultural area defined in the locale of the process, or in the locale represented by *locale*, respectively (see <langinfo.h>). The manifest constant names and values of *item* are defined in <langinfo.h>. For example:

```
nl_langinfo(ABDAY_1)
```

would return a pointer to the string "Dom" if the identified language was Portuguese, and "Sun" if the identified language was English.

```
nl_langinfo_l(ABDAY_1, loc)
```

would return a pointer to the string "Dom" if the identified language of the locale represented by *loc* was Portuguese, and "Sun" if the identified language of the locale represented by *loc* was English.

Calls to *setlocale()* with a category corresponding to the category of *item* (see <langinfo.h>), or to the category *LC_ALL*, may overwrite the array pointed to by the return value. Calls to *uselocale()* which change the category corresponding to the category of *item* may overwrite the array pointed to by the return value.

The *nl_langinfo()* function need not be thread-safe.

RETURN VALUE

In a locale where *langinfo* data is not defined, these functions shall return a pointer to the corresponding string in the POSIX locale. In all locales, these functions shall return a pointer to an empty string if *item* contains an invalid setting.

This pointer may point to static data that may be overwritten on the next call to either function.

ERRORS

The *nl_langinfo_l()* function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES**Getting Date and Time Formatting Information**

The following example returns a pointer to a string containing date and time formatting information, as defined in the *LC_TIME* category of the current locale.

```
#include <time.h>
#include <langinfo.h>
...
strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
...
```

nl_langinfo()*System Interfaces*45122 **APPLICATION USAGE**

45123 The array pointed to by the return value should not be modified by the program, but may be
45124 modified by further calls to these functions.

45125 **RATIONALE**

45126 None.

45127 **FUTURE DIRECTIONS**

45128 None.

45129 **SEE ALSO**

45130 [*setlocale\(\)*](#), [*uselocale\(\)*](#)

45131 XBD [Chapter 7](#) (on page 135), [*<langinfo.h>*](#), [*<locale.h>*](#), [*<nl_types.h>*](#)

45132 **CHANGE HISTORY**

45133 First released in Issue 2.

45134 **Issue 5**

45135 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

45136 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

45137 **Issue 7**

45138 Austin Group Interpretation 1003.1-2001 #156 is applied.

45139 The *nl_langinfo()* function is moved from the XSI option to the Base.

45140 The *nl_langinfo_l()* function is added from The Open Group Technical Standard, 2006, Extended
45141 API Set Part 4.

*System Interfaces***nrand48()**45142 **NAME**

45143 nrand48 — generate uniformly distributed pseudo-random non-negative long integers

45144 **SYNOPSIS**

```
45145 XSI      #include <stdlib.h>  
45146          long nrand48(unsigned short xsubi[3]);
```

45147 **DESCRIPTION**45148 Refer to *drand48()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

ntohl()*System Interfaces*45149 **NAME**45150 **ntohl, ntohs** — convert values between host and network byte order45151 **SYNOPSIS**45152 `#include <arpa/inet.h>`45153 `uint32_t ntohl(uint32_t netlong);`45154 `uint16_t ntohs(uint16_t netshort);`45155 **DESCRIPTION**45156 Refer to *htonl()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

open, openat — open file relative to directory file descriptor

SYNOPSIS

```
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int oflag, ...);
int openat(int fd, const char *path, int oflag, ...);
```

DESCRIPTION

The *open()* function shall establish the connection between a file and a file descriptor. It shall create an open file description that refers to a file and a file descriptor that refers to that open file description. The file descriptor is used by other I/O functions to refer to that file. The *path* argument points to a pathname naming the file.

The *open()* function shall return a file descriptor for the named file that is the lowest file descriptor not currently open for that process. The open file description is new, and therefore the file descriptor shall not share it with any other process in the system. The FD_CLOEXEC file descriptor flag associated with the new file descriptor shall be cleared unless the O_CLOEXEC flag is set in *oflag*.

The file offset used to mark the current position within the file shall be set to the beginning of the file.

The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in **<fcntl.h>**. Applications shall specify exactly one of the first five values (file access modes) below in the value of *oflag*:

O_EXEC	Open for execute only (non-directory files). The result is unspecified if this flag is applied to a directory.
O_RDONLY	Open for reading only.
O_RDWR	Open for reading and writing. The result is undefined if this flag is applied to a FIFO.
O_SEARCH	Open directory for search only. The result is unspecified if this flag is applied to a non-directory file.
O_WRONLY	Open for writing only.

Any combination of the following may be used:

O_APPEND	If set, the file offset shall be set to the end of the file prior to each write.
O_CLOEXEC	If set, the FD_CLOEXEC flag for the new file descriptor shall be set.
O_CREAT	If the file exists, this flag has no effect except as noted under O_EXCL below. Otherwise, the file shall be created; the user ID of the file shall be set to the effective user ID of the process; the group ID of the file shall be set to the group ID of the file's parent directory or to the effective group ID of the process; and the access permission bits (see <sys/stat.h>) of the file mode shall be set to the value of the argument following the <i>oflag</i> argument taken as type mode_t modified as follows: a bitwise AND is performed on the file-mode bits and the corresponding bits in the complement of the process' file mode creation mask. Thus, all bits in the

open()*System Interfaces*

45201			file mode whose corresponding bit in the file mode creation mask is set
45202			are cleared. When bits other than the file permission bits are set, the effect
45203			is unspecified. The argument following the <i>oflag</i> argument does not affect
45204			whether the file is open for reading, writing, or for both. Implementations
45205			shall provide a way to initialize the file's group ID to the group ID of the
45206			parent directory. Implementations may, but need not, provide an
45207			implementation-defined way to initialize the file's group ID to the
45208			effective group ID of the calling process.
45209		O_DIRECTORY	If <i>path</i> does not name a directory, fail and set <i>errno</i> to [ENOTDIR].
45210	SIO	O_DSYNC	Write I/O operations on the file descriptor shall complete as defined by
45211			synchronized I/O data integrity completion.
45212		O_EXCL	If O_CREAT and O_EXCL are set, <i>open()</i> shall fail if the file exists. The
45213			check for the existence of the file and the creation of the file if it does not
45214			exist shall be atomic with respect to other threads executing <i>open()</i>
45215			naming the same filename in the same directory with O_EXCL and
45216			O_CREAT set. If O_EXCL and O_CREAT are set, and <i>path</i> names a
45217			symbolic link, <i>open()</i> shall fail and set <i>errno</i> to [EEXIST], regardless of the
45218			contents of the symbolic link. If O_EXCL is set and O_CREAT is not set,
45219			the result is undefined.
45220		O_NOCTTY	If set and <i>path</i> identifies a terminal device, <i>open()</i> shall not cause the
45221			terminal device to become the controlling terminal for the process. If <i>path</i>
45222			does not identify a terminal device, O_NOCTTY shall be ignored.
45223		O_NOFOLLOW	If <i>path</i> names a symbolic link, fail and set <i>errno</i> to [ELOOP].
45224		O_NONBLOCK	When opening a FIFO with O_RDONLY or O_WRONLY set:
45225			• If O_NONBLOCK is set, an <i>open()</i> for reading-only shall return
45226			without delay. An <i>open()</i> for writing-only shall return an error if no
45227			process currently has the file open for reading.
45228			• If O_NONBLOCK is clear, an <i>open()</i> for reading-only shall block the
45229			calling thread until a thread opens the file for writing. An <i>open()</i> for
45230			writing-only shall block the calling thread until a thread opens the
45231			file for reading.
45232			When opening a block special or character special file that supports non-
45233			blocking opens:
45234			• If O_NONBLOCK is set, the <i>open()</i> function shall return without
45235			blocking for the device to be ready or available. Subsequent
45236			behavior of the device is device-specific.
45237			• If O_NONBLOCK is clear, the <i>open()</i> function shall block the calling
45238			thread until the device is ready or available before returning.
45239			Otherwise, the behavior of O_NONBLOCK is unspecified.
45240	SIO	O_RSYNC	Read I/O operations on the file descriptor shall complete at the same
45241			level of integrity as specified by the O_DSYNC and O_SYNC flags. If both
45242			O_DSYNC and O_RSYNC are set in <i>oflag</i> , all I/O operations on the file
45243			descriptor shall complete as defined by synchronized I/O data integrity
45244			completion. If both O_SYNC and O_RSYNC are set in flags, all I/O
45245			operations on the file descriptor shall complete as defined by

System Interfaces

open()

45246			synchronized I/O file integrity completion.
45247	XSI SIO	O_SYNC	Write I/O operations on the file descriptor shall complete as defined by
45248			synchronized I/O file integrity completion.
45249	XSI		The O_SYNC flag shall be supported for regular files, even if the
45250			Synchronized Input and Output option is not supported.
45251		O_TRUNC	If the file exists and is a regular file, and the file is successfully opened
45252			O_RDWR or O_WRONLY, its length shall be truncated to 0, and the mode
45253			and owner shall be unchanged. It shall have no effect on FIFO special files
45254			or terminal device files. Its effect on other file types is implementation-
45255			defined. The result of using O_TRUNC without either O_RDWR or
45256			O_WRONLY is undefined.
45257		O_TTY_INIT	If <i>path</i> identifies a terminal device other than a pseudo-terminal, the
45258			device is not already open in any process, and either O_TTY_INIT is set in
45259			<i>oflag</i> or O_TTY_INIT has the value zero, <i>open()</i> shall set any non-standard
45260			termios structure terminal parameters to a state that provides conforming
45261			behavior; see XBD Section 11.2 (on page 205). It is unspecified whether
45262			O_TTY_INIT has any effect if the device is already open in any process. If
45263			<i>path</i> identifies the slave side of a pseudo-terminal that is not already open
45264			in any process, <i>open()</i> shall set any non-standard termios structure
45265			terminal parameters to a state that provides conforming behavior,
45266			regardless of whether O_TTY_INIT is set. If <i>path</i> does not identify a
45267			terminal device, O_TTY_INIT shall be ignored.
45268			If O_CREAT is set and the file did not previously exist, upon successful completion, <i>open()</i> shall
45269			mark for update the last data access, last data modification, and last file status change
45270			timestamps of the file and the last data modification and last file status change timestamps of
45271			the parent directory.
45272			If O_TRUNC is set and the file did previously exist, upon successful completion, <i>open()</i> shall
45273			mark for update the last data modification and last file status change timestamps of the file.
45274	SIO		If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.
45275	OB XSR		If <i>path</i> refers to a STREAMS file, <i>oflag</i> may be constructed from O_NONBLOCK OR'ed with
45276			either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to STREAMS
45277			devices and shall have no effect on them. The value O_NONBLOCK affects the operation of
45278			STREAMS drivers and certain functions applied to file descriptors associated with STREAMS
45279			files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific.
45280			The application shall ensure that it specifies the O_TTY_INIT flag on the first open of a terminal
45281			device since system boot or since the device was closed by the process that last had it open. The
45282	XSI		application need not specify the O_TTY_INIT flag when opening pseudo-terminals. If <i>path</i>
45283			names the master side of a pseudo-terminal device, then it is unspecified whether <i>open()</i> locks
45284			the slave side so that it cannot be opened. Conforming applications shall call <i>unlockpt()</i> before
45285			opening the slave side.
45286			The largest value that can be represented correctly in an object of type off_t shall be established
45287			as the offset maximum in the open file description.
45288			The <i>openat()</i> function shall be equivalent to the <i>open()</i> function except in the case where <i>path</i>
45289			specifies a relative path. In this case the file to be opened is determined relative to the directory
45290			associated with the file descriptor <i>fd</i> instead of the current working directory. If the file

open()*System Interfaces*

45291 descriptor was opened without O_SEARCH, the function shall check whether directory searches
 45292 are permitted using the current permissions of the directory underlying the file descriptor. If the
 45293 file descriptor was opened with O_SEARCH, the function shall not perform the check.

45294 The *oflag* parameter and the optional fourth parameter correspond exactly to the parameters of
 45295 *open()*.

45296 If *openat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 45297 directory is used and the behavior shall be identical to a call to *open()*.

RETURN VALUE

45298 Upon successful completion, these functions shall open the file and return a non-negative
 45299 integer representing the lowest numbered unused file descriptor. Otherwise, these functions
 45300 shall return -1 and set *errno* to indicate the error. If - is returned, no files shall be created or
 45301 modified.
 45302

ERRORS

45303 These functions shall fail if:

45304 [EACCES] Search permission is denied on a component of the path prefix, or the file
 45305 exists and the permissions specified by *oflag* are denied, or the file does not
 45306 exist and write permission is denied for the parent directory of the file to be
 45307 created, or O_TRUNC is specified and write permission is denied.
 45308

45309 [EEXIST] O_CREAT and O_EXCL are set, and the named file exists.

45310 [EINTR] A signal was caught during *open()*.

45311 SIO [EINVAL] The implementation does not support synchronized I/O for this file.

45312 OB XSR [EIO] The *path* argument names a STREAMS file and a hangup or error occurred
 45313 during the *open()*.

45314 [EISDIR] The named file is a directory and *oflag* includes O_WRONLY or O_RDWR.

45315 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 45316 argument, or O_NOFOLLOW was specified and the *path* argument names a
 45317 symbolic link.

45318 [EMFILE] All file descriptors available to the process are currently open.

45319 [ENAMETOOLONG]

45320 The length of a component of a pathname is longer than {NAME_MAX}.

45321 [ENFILE] The maximum allowable number of files is currently open in the system.

45322 [ENOENT] O_CREAT is not set and the named file does not exist; or O_CREAT is set and
 45323 either the path prefix does not exist or the *path* argument points to an empty
 45324 string.

45325 OB XSR [ENOSR] The *path* argument names a STREAMS-based file and the system is unable to
 45326 allocate a STREAM.

45327 [ENOSPC] The directory or file system that would contain the new file cannot be
 45328 expanded, the file does not exist, and O_CREAT is specified.

45329 [ENOTDIR] A component of the path prefix is not a directory; or O_CREAT and O_EXCL
 45330 are not specified, the *path* argument contains at least one non-*<slash>*
 45331 character and ends with one or more trailing *<slash>* characters, and the last
 45332 pathname component names an existing file that is neither a directory nor a
 45333 symbolic link to a directory; or O_DIRECTORY was specified and the *path*

45334		argument does not name a directory.
45335	[ENXIO]	O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no
45336		process has the file open for reading.
45337	[ENXIO]	The named file is a character special or block special file, and the device
45338		associated with this special file does not exist.
45339	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented
45340		correctly in an object of type <code>off_t</code> .
45341	[EROFS]	The named file resides on a read-only file system and either O_WRONLY,
45342		O_RDWR, O_CREAT (if the file does not exist), or O_TRUNC is set in the <i>oflag</i>
45343		argument.
45344		The <i>openat()</i> function shall fail if:
45345	[EACCES]	<i>fd</i> was not opened with O_SEARCH and the permissions of the directory
45346		underlying <i>fd</i> do not permit directory searches.
45347	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is
45348		neither AT_FDCWD nor a valid file descriptor open for reading or searching.
45349		These functions may fail if:
45350	XSI [EAGAIN]	The <i>path</i> argument names the slave side of a pseudo-terminal device that is
45351		locked.
45352	[EINVAL]	The value of the <i>oflag</i> argument is not valid.
45353	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
45354		resolution of the <i>path</i> argument.
45355	[ENAMETOOLONG]	
45356		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
45357		symbolic link produced an intermediate result with a length that exceeds
45358		{PATH_MAX}.
45359	OB XSR [ENOMEM]	The <i>path</i> argument names a STREAMS file and the system is unable to allocate
45360		resources.
45361	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>oflag</i> is
45362		O_WRONLY or O_RDWR.
45363		The <i>openat()</i> function may fail if:
45364	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is neither AT_FDCWD nor a
45365		file descriptor associated with a directory.

EXAMPLES**Opening a File for Writing by the Owner**

The following example opens the file `/tmp/file`, either by creating it (if it does not already exist), or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file, the access permission bits in the file mode of the file are set to permit reading and writing by the owner, and to permit reading only by group members and others.

If the call to *open()* is successful, the file is opened for writing.

```
#include <fcntl.h>
...
```

open()

```

45375     int fd;
45376     mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
45377     char *filename = "/tmp/file";
45378     ...
45379     fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, mode);
45380     ...

```

Opening a File Using an Existence Check

The following example uses the *open()* function to try to create the **LOCKFILE** file and open it for writing. Since the *open()* function specifies the **O_EXCL** flag, the call fails if the file already exists. In that case, the program assumes that someone else is updating the password file and exits.

```

45386     #include <fcntl.h>
45387     #include <stdio.h>
45388     #include <stdlib.h>
45389
45389     #define LOCKFILE "/etc/ptmp"
45390     ...
45391     int pfd; /* Integer for file descriptor returned by open() call. */
45392     ...
45393     if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
45394         S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
45395     {
45396         fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
45397         exit(1);
45398     }
45399     ...

```

Opening a File for Writing

The following example opens a file for writing, creating the file if it does not already exist. If the file does exist, the system truncates the file to zero bytes.

```

45403     #include <fcntl.h>
45404     #include <stdio.h>
45405     #include <stdlib.h>
45406
45406     #define LOCKFILE "/etc/ptmp"
45407     ...
45408     int pfd;
45409     char filename[PATH_MAX+1];
45410     ...
45411     if ((pfd = open(filename, O_WRONLY | O_CREAT | O_TRUNC,
45412         S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
45413     {
45414         perror("Cannot open output file\n"); exit(1);
45415     }
45416     ...

```

APPLICATION USAGE

POSIX.1-2008 does not require that terminal parameters be automatically set to any state on first open, nor that they be reset after the last close. It is possible for a non-conforming application to leave a terminal device in a state where the next process to use that device finds it in a non-conforming state, but has no way of determining this. To ensure that the device is set to a conforming initial state, applications which perform a first open of a terminal (other than a pseudo-terminal) should do so using the `O_TTY_INIT` flag to set the parameters associated with the terminal to a conforming state.

RATIONALE

Except as specified in this volume of POSIX.1-2008, the flags allowed in *oflag* are not mutually-exclusive and any number of them may be used simultaneously.

Some implementations permit opening FIFOs with `O_RDWR`. Since FIFOs could be implemented in other ways, and since two file descriptors can be used to the same effect, this possibility is left as undefined.

See `getgroups()` about the group of a newly created file.

The use of `open()` to create a regular file is preferable to the use of `creat()`, because the latter is redundant and included only for historical reasons.

The use of the `O_TRUNC` flag on FIFOs and directories (pipes cannot be `open()`-ed) must be permissible without unexpected side-effects (for example, `creat()` on a FIFO must not remove data). Since terminal special files might have type-ahead data stored in the buffer, `O_TRUNC` should not affect their content, particularly if a program that normally opens a regular file should open the current controlling terminal instead. Other file types, particularly implementation-defined ones, are left implementation-defined.

POSIX.1-2008 permits `[EACCES]` to be returned for conditions other than those explicitly listed.

The `O_NOCTTY` flag was added to allow applications to avoid unintentionally acquiring a controlling terminal as a side-effect of opening a terminal file. This volume of POSIX.1-2008 does not specify how a controlling terminal is acquired, but it allows an implementation to provide this on `open()` if the `O_NOCTTY` flag is not set and other conditions specified in XBD Chapter 11 (on page 199) are met.

In historical implementations the value of `O_RDONLY` is zero. Because of that, it is not possible to detect the presence of `O_RDONLY` and another option. Future implementations should encode `O_RDONLY` and `O_WRONLY` as bit flags so that:

```
O_RDONLY | O_WRONLY == O_RDWR
```

`O_EXEC` and `O_SEARCH` are specified as two of the five file access modes. Since `O_EXEC` does not apply to directories, and `O_SEARCH` only applies to directories, their values need not be distinct. Since `O_RDONLY` has historically had the value zero, implementations are not able to distinguish between `O_SEARCH` and `O_SEARCH | O_RDONLY`, and similarly for `O_EXEC`.

In general, the `open()` function follows the symbolic link if *path* names a symbolic link. However, the `open()` function, when called with `O_CREAT` and `O_EXCL`, is required to fail with `[EEXIST]` if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This behavior is required so that privileged applications can create a new file in a known location without the possibility that a symbolic link might cause the file to be created in a different location.

For example, a privileged application that must create a file with a predictable name in a user-writable directory, such as the user's home directory, could be compromised if the user creates a symbolic link with that name that refers to a nonexistent file in a system directory. If the user can

open()*System Interfaces*

influence the contents of a file, the user could compromise the system by creating a new system configuration or spool file that would then be interpreted by the system. The test for a symbolic link which refers to a nonexisting file must be atomic with the creation of a new file.

In addition, the *open()* function refuses to open non-directories if the *O_DIRECTORY* flag is set. This avoids race conditions whereby a user might compromise the system by substituting a hard link to a sensitive file (e.g., a device or a FIFO) while a privileged application is running, where opening a file even for read access might have undesirable side-effects.

In addition, the *open()* function does not follow symbolic links if the *O_NOFOLLOW* flag is set. This avoids race conditions whereby a user might compromise the system by substituting a symbolic link to a sensitive file (e.g., a device) while a privileged application is running, where opening a file even for read access might have undesirable side-effects.

The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown()* to set the group ID after the file is created, or determine under what conditions the implementation will set the desired group ID.

The purpose of the *openat()* function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *open()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *openat()* function it can be guaranteed that the opened file is located relative to the desired directory. Some implementations use the *openat()* function for other purposes as well. In some cases, if the *oflag* parameter has the *O_XATTR* bit set, the returned file descriptor provides access to extended attributes. This functionality is not standardized here.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod(), *close()*, *creat()*, *dirfd()*, *dup()*, *exec*, *fcntl()*, *fdopendir()*, *link()*, *lseek()*, *mkdtemp()*, *mknod()*, *read()*, *symlink()*, *umask()*, *unlockpt()*, *write()*

XBD Chapter 11 (on page 199), [*<fcntl.h>*](#), [*<sys/stat.h>*](#), [*<sys/types.h>*](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

Issue 6

In the SYNOPSIS, the optional include of the [*<sys/types.h>*](#) header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the DESCRIPTION, `O_CREAT` is amended to state that the group ID of the file is set to the group ID of the file's parent directory or to the effective group ID of the process. This is a FIPS requirement.
- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the `[EOVERFLOW]` condition is added. This change is to support large files.
- The `[ENXIO]` mandatory error condition is added.
- The `[EINVAL]`, `[ENAMETOOLONG]`, and `[ETXTBSY]` optional error conditions are added.

The DESCRIPTION and ERRORS sections are updated so that items related to the optional XSI STREAMS Option Group are marked.

The following changes were made to align with the IEEE P1003.1a draft standard:

- An explanation is added of the effect of the `O_CREAT` and `O_EXCL` flags when the path refers to a symbolic link.
- The `[ELOOP]` optional error condition is added.

The normative text is updated to avoid use of the term “must” for application requirements.

The DESCRIPTION of `O_EXCL` is updated in response to IEEE PASC Interpretation 1003.1c #48.

Issue 7

Austin Group Interpretations 1003.1-2001 #113 and #143 are applied.

Austin Group Interpretation 1003.1-2001 #144 is applied, adding the `O_TTY_INIT` flag.

Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the `FD_CLOEXEC` flag atomically at `open()`, and adding the `F_DUPFD_CLOEXEC` flag.

SD5-XBD-ERN-4 is applied, changing the definition of the `[EMFILE]` error.

This page is revised and the `openat()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

open_memstream()

System Interfaces

45537 **NAME**45538 `open_memstream`, `open_wmemstream` — open a dynamic memory buffer stream45539 **SYNOPSIS**

```

45540 CX      #include <stdio.h>
45541          FILE *open_memstream(char **bufp, size_t *sizep);
45542          #include <wchar.h>
45543          FILE *open_wmemstream(wchar_t **bufp, size_t *sizep);

```

45544 **DESCRIPTION**

45545 The `open_memstream()` and `open_wmemstream()` functions shall create an I/O stream associated
 45546 with a dynamically allocated memory buffer. The stream shall be opened for writing and shall
 45547 be seekable.

45548 The stream associated with a call to `open_memstream()` shall be byte-oriented.

45549 The stream associated with a call to `open_wmemstream()` shall be wide-oriented.

45550 The stream shall maintain a current position in the allocated buffer and a current buffer length.
 45551 The position shall be initially set to zero (the start of the buffer). Each write to the stream shall
 45552 start at the current position and move this position by the number of successfully written bytes
 45553 for `open_memstream()` or the number of successfully written wide characters for
 45554 `open_wmemstream()`. The length shall be initially set to zero. If a write moves the position to a
 45555 value larger than the current length, the current length shall be set to this position. In this case a
 45556 null character for `open_memstream()` or a null wide character for `open_wmemstream()` shall be
 45557 appended to the current buffer. For both functions the terminating null is not included in the
 45558 calculation of the buffer length.

45559 After a successful `fflush()` or `fclose()`, the pointer referenced by `bufp` shall contain the address of
 45560 the buffer, and the variable pointed to by `sizep` shall contain the smaller of the current buffer
 45561 length and the number of bytes for `open_memstream()`, or the number of wide characters for
 45562 `open_wmemstream()`, between the beginning of the buffer and the current file position indicator.

45563 After a successful `fflush()` the pointer referenced by `bufp` and the variable referenced by `sizep`
 45564 remain valid only until the next write operation on the stream or a call to `fclose()`.

45565 **RETURN VALUE**

45566 Upon successful completion, these functions shall return a pointer to the object controlling the
 45567 stream. Otherwise, a null pointer shall be returned, and `errno` shall be set to indicate the error.

45568 **ERRORS**

45569 These functions may fail if:

45570	[EINVAL]	<code>bufp</code> or <code>sizep</code> are NULL.
45571	[EMFILE]	{FOPEN_MAX} streams are currently open in the calling process.
45572	[ENOMEM]	Memory for the stream or the buffer could not be allocated.

EXAMPLES

```

#include <stdio.h>
#include <stdlib.h>

int
main (void)
{
    FILE *stream;
    char *buf;
    size_t len;
    off_t eob;

    stream = open_memstream (&buf, &len);
    if (stream == NULL)
        /* handle error */ ;
    fprintf (stream, "hello my world");
    fflush (stream);
    printf ("buf=%s, len=%zu\n", buf, len);
    eob = ftello(stream);
    fseeko (stream, 0, SEEK_SET);
    fprintf (stream, "good-bye");
    fseeko (stream, eob, SEEK_SET);
    fclose (stream);
    printf ("buf=%s, len=%zu\n", buf, len);
    free (buf);
    return 0;
}

```

This program produces the following output:

```

buf=hello my world, len=14
buf=good-bye world, len=14

```

APPLICATION USAGE

The buffer created by these functions should be freed by the application after closing the stream, by means of a call to *free()*.

RATIONALE

These functions are similar to *fmemopen()* except that the memory is always allocated dynamically by the function, and the stream is opened only for output.

FUTURE DIRECTIONS

None.

SEE ALSO

fclose(), *fdopen()*, *fflush()*, *fmemopen()*, *fopen()*, *free()*, *freopen()*

XBD *<stdio.h>*, *<wchar.h>*

CHANGE HISTORY

First released in Issue 7.

openat()*System Interfaces*45614 **NAME**

45615 openat — open file relative to directory file descriptor

45616 **SYNOPSIS**

45617 #include <fcntl.h>

45618 int openat(int *fd*, const char **path*, int *oflag*, ...);45619 **DESCRIPTION**45620 Refer to *open()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***opendir()**45621 **NAME**

45622 opendir — open directory associated with file descriptor

45623 **SYNOPSIS**

45624 #include <dirent.h>

45625 DIR *opendir(const char *dirname);

45626 **DESCRIPTION**45627 Refer to *fdopendir()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

openlog()*System Interfaces*45628 **NAME**

45629 openlog — open a connection to the logging facility

45630 **SYNOPSIS**

45631 XSI #include <syslog.h>

45632 void openlog(const char *ident, int logopt, int facility);

45633 **DESCRIPTION**45634 Refer to *closelog()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***optarg**45635 **NAME**

45636 optarg, opterr, optind, optopt — options parsing variables

45637 **SYNOPSIS**

45638 #include <unistd.h>

45639 extern char *optarg;

45640 extern int opterr, optind, optopt;

45641 **DESCRIPTION**45642 Refer to *getopt()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pathconf()*System Interfaces*45643 **NAME**45644 `pathconf` — get configurable pathname variables45645 **SYNOPSIS**45646 `#include <unistd.h>`45647 `long pathconf(const char *path, int name);`45648 **DESCRIPTION**45649 Refer to *fpathconf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

pause — suspend the thread until a signal is received

SYNOPSIS

```
#include <unistd.h>
```

```
int pause(void);
```

DESCRIPTION

The *pause()* function shall suspend the calling thread until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process.

If the action is to terminate the process, *pause()* shall not return.

If the action is to execute a signal-catching function, *pause()* shall return after the signal-catching function returns.

RETURN VALUE

Since *pause()* suspends thread execution indefinitely unless interrupted by a signal, there is no successful completion return value. A value of -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *pause()* function shall fail if:

[EINTR]	A signal is caught by the calling process and control is returned from the signal-catching function.

EXAMPLES

None.

APPLICATION USAGE

Many common uses of *pause()* have timing windows. The scenario involves checking a condition related to a signal and, if the signal has not occurred, calling *pause()*. When the signal occurs between the check and the call to *pause()*, the process often blocks indefinitely. The *sigprocmask()* and *sigsuspend()* functions can be used to avoid this type of problem.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sigsuspend()

XBD *<unistd.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The APPLICATION USAGE section is added.

pclose()*System Interfaces*45689 **NAME**45690 **pclose** — close a pipe stream to or from a process45691 **SYNOPSIS**

```
45692 CX #include <stdio.h>
45693 int pclose(FILE *stream);
```

45694 **DESCRIPTION**

45695 The *pclose()* function shall close a stream that was opened by *popen()*, wait for the command to
 45696 terminate, and return the termination status of the process that was running the command
 45697 language interpreter. However, if a call caused the termination status to be unavailable to
 45698 *pclose()*, then *pclose()* shall return -1 with *errno* set to [ECHILD] to report this situation. This can
 45699 happen if the application calls one of the following functions:

- 45700 • *wait()*
- 45701 • *waitpid()* with a *pid* argument less than or equal to 0 or equal to the process ID of the
 45702 command line interpreter
- 45703 • Any other function not defined in this volume of POSIX.1-2008 that could do one of the
 45704 above

45705 In any case, *pclose()* shall not return before the child process created by *popen()* has terminated.

45706 If the command language interpreter cannot be executed, the child termination status returned
 45707 by *pclose()* shall be as if the command language interpreter terminated using *exit(127)* or
 45708 *_exit(127)*.

45709 The *pclose()* function shall not affect the termination status of any child of the calling process
 45710 other than the one created by *popen()* for the associated stream.

45711 If the argument *stream* to *pclose()* is not a pointer to a stream created by *popen()*, the result of
 45712 *pclose()* is undefined.

45713 **RETURN VALUE**

45714 Upon successful return, *pclose()* shall return the termination status of the command language
 45715 interpreter. Otherwise, *pclose()* shall return -1 and set *errno* to indicate the error.

45716 **ERRORS**

45717 The *pclose()* function shall fail if:

- 45718 [ECHILD] The status of the child process could not be obtained, as described above.

45719 **EXAMPLES**

45720 None.

45721 **APPLICATION USAGE**

45722 None.

45723 **RATIONALE**

45724 There is a requirement that *pclose()* not return before the child process terminates. This is
 45725 intended to disallow implementations that return [EINTR] if a signal is received while waiting.
 45726 If *pclose()* returned before the child terminated, there would be no way for the application to
 45727 discover which child used to be associated with the stream, and it could not do the cleanup
 45728 itself.

45729 If the stream pointed to by *stream* was not created by *popen()*, historical implementations of
 45730 *pclose()* return -1 without setting *errno*. To avoid requiring *pclose()* to set *errno* in this case,
 45731 POSIX.1-2008 makes the behavior unspecified. An application should not use *pclose()* to close

any stream that was not created by *popen()*.

Some historical implementations of *pclose()* either block or ignore the signals SIGINT, SIGQUIT, and SIGHUP while waiting for the child process to terminate. Since this behavior is not described for the *pclose()* function in POSIX.1-2008, such implementations are not conforming. Also, some historical implementations return [EINTR] if a signal is received, even though the child process has not terminated. Such implementations are also considered non-conforming.

Consider, for example, an application that uses:

```
popen("command", "r")
```

to start *command*, which is part of the same application. The parent writes a prompt to its standard output (presumably the terminal) and then reads from the *popen()*ed stream. The child reads the response from the user, does some transformation on the response (pathname expansion, perhaps) and writes the result to its standard output. The parent process reads the result from the pipe, does something with it, and prints another prompt. The cycle repeats. Assuming that both processes do appropriate buffer flushing, this would be expected to work.

To conform to POSIX.1-2008, *pclose()* must use *waitpid()*, or some similar function, instead of *wait()*.

The code sample below illustrates how the *pclose()* function might be implemented on a system conforming to POSIX.1-2008.

```
int pclose(FILE *stream)
{
    int stat;
    pid_t pid;

    pid = <pid for process created for stream by popen(>
    (void) fclose(stream);
    while (waitpid(pid, &stat, 0) == -1) {
        if (errno != EINTR) {
            stat = -1;
            break;
        }
    }
    return(stat);
}
```

FUTURE DIRECTIONS

None.

SEE ALSO

fork(), *popen()*, *wait()*

XBD <stdio.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

perror()*System Interfaces*45771 **NAME**45772 **perror** — write error messages to standard error45773 **SYNOPSIS**45774 `#include <stdio.h>`45775 `void perror(const char *s);`45776 **DESCRIPTION**

45777 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 45778 conflict between the requirements described here and the ISO C standard is unintentional. This
 45779 volume of POSIX.1-2008 defers to the ISO C standard.

45780 The *perror()* function shall map the error number accessed through the symbol *errno* to a
 45781 language-dependent error message, which shall be written to the standard error stream as
 45782 follows:

- 45783 • First (if *s* is not a null pointer and the character pointed to by *s* is not the null byte), the
 45784 string pointed to by *s* followed by a <colon> and a <space>.
- 45785 • Then an error message string followed by a <newline>.

45786 The contents of the error message strings shall be the same as those returned by *strerror()* with
 45787 argument *errno*.

45788 CX The *perror()* function shall mark for update the last data modification and last file status change
 45789 timestamps of the file associated with the standard error stream at some time between its
 45790 successful completion and *exit()*, *abort()*, or the completion of *fflush()* or *fclose()* on *stderr*.

45791 The *perror()* function shall not change the orientation of the standard error stream.

45792 **RETURN VALUE**45793 The *perror()* function shall not return a value.45794 **ERRORS**

45795 No errors are defined.

45796 **EXAMPLES**45797 **Printing an Error Message for a Function**

45798 The following example replaces *bufptr* with a buffer that is the necessary size. If an error occurs,
 45799 the *perror()* function prints a message and the program exits.

```

45800     #include <stdio.h>
45801     #include <stdlib.h>
45802     ...
45803     char *bufptr;
45804     size_t szbuf;
45805     ...
45806     if ((bufptr = malloc(szbuf)) == NULL) {
45807         perror("malloc"); exit(2);
45808     }
45809     ...
  
```

45810 **APPLICATION USAGE**

45811 None.

45812 **RATIONALE**

45813 None.

45814 **FUTURE DIRECTIONS**

45815 None.

45816 **SEE ALSO**45817 *psiginfo()*, *strerror()*45818 XBD *<stdio.h>*45819 **CHANGE HISTORY**

45820 First released in Issue 1. Derived from Issue 1 of the SVID.

45821 **Issue 5**45822 A paragraph is added to the DESCRIPTION indicating that *perror()* does not change the
45823 orientation of the standard error stream.45824 **Issue 6**

45825 Extensions beyond the ISO C standard are marked.

45826 **Issue 7**

45827 SD5-XSH-ERN-95 is applied.

45828 Changes are made related to support for finegrained timestamps.

pipe()

System Interfaces

45829 **NAME**

45830 pipe — create an interprocess channel

45831 **SYNOPSIS**

45832 #include <unistd.h>

45833 int pipe(int *fildes*[2]);45834 **DESCRIPTION**

45835 The *pipe()* function shall create a pipe and place two file descriptors, one each into the
 45836 arguments *fildes*[0] and *fildes*[1], that refer to the open file descriptions for the read and write
 45837 ends of the pipe. Their integer values shall be the two lowest available at the time of the *pipe()*
 45838 call. The O_NONBLOCK and FD_CLOEXEC flags shall be clear on both file descriptors. (The
 45839 *fcntl()* function can be used to set both these flags.)

45840 Data can be written to the file descriptor *fildes*[1] and read from the file descriptor *fildes*[0]. A
 45841 read on the file descriptor *fildes*[0] shall access data written to the file descriptor *fildes*[1] on a
 45842 first-in-first-out basis. It is unspecified whether *fildes*[0] is also open for writing and whether
 45843 *fildes*[1] is also open for reading.

45844 A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open
 45845 that refers to the read end, *fildes*[0] (write end, *fildes*[1]).

45846 The pipe's user ID shall be set to the effective user ID of the calling process.

45847 The pipe's group ID shall be set to the effective group ID of the calling process.

45848 Upon successful completion, *pipe()* shall mark for update the last data access, last data
 45849 modification, and last file status change timestamps of the pipe.

45850 **RETURN VALUE**

45851 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 45852 indicate the error.

45853 **ERRORS**

45854 The *pipe()* function shall fail if:

45855	[EMFILE]	All, or all but one, of the file descriptors available to the process are currently open.
45857	[ENFILE]	The number of simultaneously open files in the system would exceed a system-imposed limit.

45859 **EXAMPLES**45860 **Using a Pipe to Pass Data Between a Parent Process and a Child Process**

45861 The following example demonstrates the use of a pipe to transfer data between a parent process
 45862 and a child process. Error handling is excluded, but otherwise this code demonstrates good
 45863 practice when using pipes: after the *fork()* the two processes close the unused ends of the pipe
 45864 before they commence transferring data.

45865 #include <stdlib.h>

45866 #include <unistd.h>

45867 ...

45868 int fildes[2];

45869 const int BSIZE = 100;

45870 char buf[BSIZE];

45871 ssize_t nbytes;

```

45872     int status;
45873     status = pipe(fildes);
45874     if (status == -1 ) {
45875         /* an error occurred */
45876         ...
45877     }
45878     switch (fork()) {
45879     case -1: /* Handle error */
45880         break;
45881     case 0: /* Child - reads from pipe */
45882         close(fildes[1]); /* Write end is unused */
45883         nbytes = read(fildes[0], buf, BSIZE); /* Get data from pipe */
45884         /* At this point, a further read would see end of file ... */
45885         close(fildes[0]); /* Finished with pipe */
45886         exit(EXIT_SUCCESS);
45887     default: /* Parent - writes to pipe */
45888         close(fildes[0]); /* Read end is unused */
45889         write(fildes[1], "Hello world\n", 12); /* Write data on pipe */
45890         close(fildes[1]); /* Child will see EOF */
45891         exit(EXIT_SUCCESS);
45892     }

```

APPLICATION USAGE

None.

RATIONALE

The wording carefully avoids using the verb “to open” in order to avoid any implication of use of *open()*; see also *write()*.

FUTURE DIRECTIONS

None.

SEE ALSO

fcntl(), *read()*, *write()*
 XBD <fcntl.h>, <unistd.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to indicate that certain dispositions of *fildes[0]* and *fildes[1]* are unspecified.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/65 is applied, adding the example to the EXAMPLES section.

pipe()*System Interfaces*45912 **Issue 7**

45913 SD5-XSH-ERN-156 is applied, updating the DESCRIPTION to state the setting of the pipe's user
45914 ID and group ID.

45915 Changes are made related to support for finegrained timestamps.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

poll()**NAME**

poll — input/output multiplexing

SYNOPSIS

#include <poll.h>

int poll(struct pollfd *fds*[], nfds_t *nfds*, int *timeout*);**DESCRIPTION**

The *poll()* function provides applications with a mechanism for multiplexing input/output over a set of file descriptors. For each member of the array pointed to by *fds*, *poll()* shall examine the given file descriptor for the event(s) specified in *events*. The number of **pollfd** structures in the *fds* array is specified by *nfds*. The *poll()* function shall identify those file descriptors on which an application can read or write data, or on which certain events have occurred.

The *fds* argument specifies the file descriptors to be examined and the events of interest for each file descriptor. It is a pointer to an array with one member for each open file descriptor of interest. The array's members are **pollfd** structures within which *fd* specifies an open file descriptor and *events* and *revents* are bitmasks constructed by OR'ing a combination of the following event flags:

POLLIN Data other than high-priority data may be read without blocking.

OB XSR

For STREAMS, this flag is set in *revents* even if the message is of zero length. This flag shall be equivalent to **POLLRDNORM** | **POLLRDBAND**.

POLLRDNORM Normal data may be read without blocking.

OB XSR

For STREAMS, data on priority band 0 may be read without blocking. This flag is set in *revents* even if the message is of zero length.

POLLRDBAND Priority data may be read without blocking.

OB XSR

For STREAMS, data on priority bands greater than 0 may be read without blocking. This flag is set in *revents* even if the message is of zero length.

POLLPRI High-priority data may be read without blocking.

OB XSR

For STREAMS, this flag is set in *revents* even if the message is of zero length.

POLLOUT Normal data may be written without blocking.

OB XSR

For STREAMS, data on priority band 0 may be written without blocking.

POLLWRNORM Equivalent to **POLLOUT**.

POLLWRBAND Priority data may be written.

OB XSR

For STREAMS, data on priority bands greater than 0 may be written without blocking. If any priority band has been written to on this STREAM, this event only examines bands that have been written to at least once.

POLLERR

An error has occurred on the device or stream. This flag is only valid in the *revents* bitmask; it shall be ignored in the *events* member.

POLLHUP

A device has been disconnected, or a pipe or FIFO has been closed by the last process that had it open for writing. Once set, the hangup state of a FIFO shall persist until some process opens the FIFO for writing or until all read-only file descriptors for the FIFO are closed. This event and **POLLOUT** are mutually-exclusive; a stream can never be writable if a hangup has occurred. However, this event and **POLLIN**, **POLLRDNORM**, **POLLRDBAND**, or **POLLPRI** are not mutually-exclusive. This flag is only valid in the *revents* bitmask; it shall be

poll()*System Interfaces*

45959		ignored in the <i>events</i> member.
45960	POLLNVAL	The specified <i>fd</i> value is invalid. This flag is only valid in the <i>revents</i> member;
45961		it shall ignored in the <i>events</i> member.
45962		The significance and semantics of normal, priority, and high-priority data are file and device-
45963		specific.
45964		If the value of <i>fd</i> is less than 0, <i>events</i> shall be ignored, and <i>revents</i> shall be set to 0 in that entry on
45965		return from <i>poll()</i> .
45966		In each pollfd structure, <i>poll()</i> shall clear the <i>revents</i> member, except that where the application
45967		requested a report on a condition by setting one of the bits of <i>events</i> listed above, <i>poll()</i> shall set
45968		the corresponding bit in <i>revents</i> if the requested condition is true. In addition, <i>poll()</i> shall set the
45969		POLLHUP, POLLERR, and POLLNVAL flag in <i>revents</i> if the condition is true, even if the
45970		application did not set the corresponding bit in <i>events</i> .
45971		If none of the defined events have occurred on any selected file descriptor, <i>poll()</i> shall wait at
45972		least <i>timeout</i> milliseconds for an event to occur on any of the selected file descriptors. If the value
45973		of <i>timeout</i> is 0, <i>poll()</i> shall return immediately. If the value of <i>timeout</i> is -1, <i>poll()</i> shall block until
45974		a requested event occurs or until the call is interrupted.
45975		Implementations may place limitations on the granularity of timeout intervals. If the requested
45976		timeout interval requires a finer granularity than the implementation supports, the actual
45977		timeout interval shall be rounded up to the next supported value.
45978		The <i>poll()</i> function shall not be affected by the O_NONBLOCK flag.
45979		The <i>poll()</i> function shall support regular files, terminal and pseudo-terminal devices, FIFOs,
45980	OB XSR	pipes, sockets and STREAMS-based files . The behavior of <i>poll()</i> on elements of <i>fds</i> that refer to
45981		other types of file is unspecified.
45982		Regular files shall always poll TRUE for reading and writing.
45983		A file descriptor for a socket that is listening for connections shall indicate that it is ready for
45984		reading, once connections are available. A file descriptor for a socket that is connecting
45985		asynchronously shall indicate that it is ready for writing, once a connection has been established.
45986	RETURN VALUE	
45987		Upon successful completion, <i>poll()</i> shall return a non-negative value. A positive value indicates
45988		the total number of file descriptors that have been selected (that is, file descriptors for which the
45989		<i>revents</i> member is non-zero). A value of 0 indicates that the call timed out and no file descriptors
45990		have been selected. Upon failure, <i>poll()</i> shall return -1 and set <i>errno</i> to indicate the error.
45991	ERRORS	
45992		The <i>poll()</i> function shall fail if:
45993	[EAGAIN]	The allocation of internal data structures failed but a subsequent request may
45994		succeed.
45995	[EINTR]	A signal was caught during <i>poll()</i> .
45996	OB XSR [EINVAL]	The <i>nfds</i> argument is greater than {OPEN_MAX}, or one of the <i>fd</i> members
45997		refers to a STREAM or multiplexer that is linked (directly or indirectly)
45998		downstream from a multiplexer.

EXAMPLES

Checking for Events on a Stream

The following example opens a pair of STREAMS devices and then waits for either one to become writable. This example proceeds as follows:

1. Sets the *timeout* parameter to 500 milliseconds.
2. Opens the STREAMS devices */dev/dev0* and */dev/dev1*, and then polls them, specifying POLLOUT and POLLWRBAND as the events of interest.

The STREAMS device names */dev/dev0* and */dev/dev1* are only examples of how STREAMS devices can be named; STREAMS naming conventions may vary among systems conforming to the POSIX.1-2008.

3. Uses the *ret* variable to determine whether an event has occurred on either of the two STREAMS. The *poll()* function is given 500 milliseconds to wait for an event to occur (if it has not occurred prior to the *poll()* call).
4. Checks the returned value of *ret*. If a positive value is returned, one of the following can be done:
 - a. Priority data can be written to the open STREAM on priority bands greater than 0, because the POLLWRBAND event occurred on the open STREAM (*fds[0]* or *fds[1]*).
 - b. Data can be written to the open STREAM on priority-band 0, because the POLLOUT event occurred on the open STREAM (*fds[0]* or *fds[1]*).
5. If the returned value is not a positive value, permission to write data to the open STREAM (on any priority band) is denied.
6. If the POLLHUP event occurs on the open STREAM (*fds[0]* or *fds[1]*), the device on the open STREAM has disconnected.

```
#include <stropts.h>
#include <poll.h>
...
struct pollfd fds[2];
int timeout_msecs = 500;
int ret;
int i;

/* Open STREAMS device. */
fds[0].fd = open("/dev/dev0", ...);
fds[1].fd = open("/dev/dev1", ...);
fds[0].events = POLLOUT | POLLWRBAND;
fds[1].events = POLLOUT | POLLWRBAND;

ret = poll(fds, 2, timeout_msecs);

if (ret > 0) {
    /* An event on one of the fds has occurred. */
    for (i=0; i<2; i++) {
        if (fds[i].revents & POLLWRBAND) {
            /* Priority data may be written on device number i. */
            ...
        }
        if (fds[i].revents & POLLOUT) {
```

poll()*System Interfaces*

```

46043             /* Data may be written on device number i. */
46044     ...
46045     }
46046     if (fds[i].revents & POLLHUP) {
46047         /* A hangup has occurred on device number i. */
46048     ...
46049     }
46050     }
46051 }

```

APPLICATION USAGE

None.

RATIONALE

The POLLHUP event does not occur for FIFOs just because the FIFO is not open for writing. It only occurs when the FIFO is closed by the last writer and persists until some process opens the FIFO for writing or until all read-only file descriptors for the FIFO are closed.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6 (on page 494), *getmsg()*, *pselect()*, *putmsg()*, *read()*, *write()*

XBD **<poll.h>**, **<stropts.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The description of POLLWRBAND is updated.

Issue 6

Text referring to sockets is added to the DESCRIPTION.

Functionality relating to the XSI STREAMS Option Group is marked.

The Open Group Corrigendum U055/3 is applied, updating the DESCRIPTION of POLLWRBAND.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/66 is applied, correcting the spacing in the EXAMPLES section.

Issue 7

Austin Group Interpretation 1003.1-2001 #209 is applied, clarifying the POLLHUP event.

The *poll()* function is moved from the XSI option to the Base.

Functionality relating to the XSI STREAMS option is marked obsolescent.

System Interfaces

popen()**NAME**

popen — initiate pipe streams to or from a process

SYNOPSIS

```
CX      #include <stdio.h>
46083   FILE *popen(const char *command, const char *mode);
```

DESCRIPTION

The *popen()* function shall execute the command specified by the string *command*. It shall create a pipe between the calling program and the executed command, and shall return a pointer to a stream that can be used to either read from or write to the pipe.

The environment of the executed command shall be as if a child process were created within the *popen()* call using the *fork()* function, and the child invoked the *sh* utility using the call:

```
execl(shell path, "sh", "-c", command, (char *)0);
```

where *shell path* is an unspecified pathname for the *sh* utility.

The *popen()* function shall ensure that any streams from previous *popen()* calls that remain open in the parent process are closed in the new child process.

The *mode* argument to *popen()* is a string that specifies I/O mode:

1. If *mode* is *r*, when the child process is started, its file descriptor *STDOUT_FILENO* shall be the writable end of the pipe, and the file descriptor *fileno(stream)* in the calling process, where *stream* is the stream pointer returned by *popen()*, shall be the readable end of the pipe.
2. If *mode* is *w*, when the child process is started its file descriptor *STDIN_FILENO* shall be the readable end of the pipe, and the file descriptor *fileno(stream)* in the calling process, where *stream* is the stream pointer returned by *popen()*, shall be the writable end of the pipe.
3. If *mode* is any other value, the result is unspecified.

After *popen()*, both the parent and the child process shall be capable of executing independently before either terminates.

Pipe streams are byte-oriented.

RETURN VALUE

Upon successful completion, *popen()* shall return a pointer to an open stream that can be used to read or write to the pipe. Otherwise, it shall return a null pointer and may set *errno* to indicate the error.

ERRORS

The *popen()* function shall fail if:

[EMFILE] {STREAM_MAX} streams are currently open in the calling process.

The *popen()* function may fail if:

[EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

[EINVAL] The *mode* argument is invalid.

The *popen()* function may also set *errno* values as described by *fork()* or *pipe()*.

popen()46118 **EXAMPLES**46119 **Using popen() to Obtain a List of Files from the ls Utility**

46120 The following example demonstrates the use of *popen()* and *pclose()* to execute the command *ls**
 46121 in order to obtain a list of files in the current directory:

```

46122 #include <stdio.h>
46123 ...
46124 FILE *fp;
46125 int status;
46126 char path[PATH_MAX];
46127 fp = popen("ls *", "r");
46128 if (fp == NULL)
46129     /* Handle error */;
46130 while (fgets(path, PATH_MAX, fp) != NULL)
46131     printf("%s", path);
46132 status = pclose(fp);
46133 if (status == -1) {
46134     /* Error reported by pclose() */
46135     ...
46136 } else {
46137     /* Use macros described under wait() to inspect 'status' in order
46138        to determine success/failure of command executed by popen() */
46139     ...
46140 }

```

46141 **APPLICATION USAGE**

46142 Since open files are shared, a mode *r* command can be used as an input filter and a mode *w*
 46143 command as an output filter.

46144 Buffered reading before opening an input filter may leave the standard input of that filter
 46145 mispositioned. Similar problems with an output filter may be prevented by careful buffer
 46146 flushing; for example, with *fflush()*.

46147 A stream opened by *popen()* should be closed by *pclose()*.

46148 The behavior of *popen()* is specified for values of *mode* of *r* and *w*. Other modes such as *rb* and
 46149 *wb* might be supported by specific implementations, but these would not be portable features.
 46150 Note that historical implementations of *popen()* only check to see if the first character of *mode* is
 46151 *r*. Thus, a mode of *robert the robot* would be treated as mode *r*, and a mode of *anything else* would be
 46152 treated as mode *w*.

46153 If the application calls *waitpid()* or *waitid()* with a *pid* argument greater than 0, and it still has a
 46154 stream that was called with *popen()* open, it must ensure that *pid* does not refer to the process
 46155 started by *popen()*.

46156 To determine whether or not the environment specified in the Shell and Utilities volume of
 46157 POSIX.1-2008 is present, use the function call:

```
46158 sysconf(_SC_2_VERSION)
```

46159 (See *sysconf()*).

RATIONALE

The *popen()* function should not be used by programs that have set user (or group) ID privileges. The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used instead. This prevents any unforeseen manipulation of the environment of the user that could cause execution of commands not anticipated by the calling program.

If the original and *popen()*ed processes both intend to read or write or read and write a common file, and either will be using FILE-type C functions (*fread()*, *fwrite()*, and so on), the rules for sharing file handles must be observed (see [Section 2.5.1](#), on page 491).

FUTURE DIRECTIONS

None.

SEE ALSO

fork(), *pclose()*, *pipe()*, *sysconf()*, *system()*, *wait()*, *waitid()*

XBD [<stdio.h>](#)

XCU *sh*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A statement is added to the DESCRIPTION indicating that pipe streams are byte-oriented.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional [EMFILE] error condition is added.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/67 is applied, adding the example to the EXAMPLES section.

Issue 7

Austin Group Interpretation 1003.1-2001 #029 is applied, clarifying the values for *mode* in the DESCRIPTION.

SD5-XSH-ERN-149 is applied, changing the {STREAM_MAX} [EMFILE] error condition from a "may fail" to a "shall fail".

posix_fadvise()

System Interfaces

46189 **NAME**46190 **posix_fadvise** — file advisory information (**ADVANCED REALTIME**)46191 **SYNOPSIS**

```
46192 ADV #include <fcntl.h>
46193 int posix_fadvise(int fd, off_t offset, off_t len, int advice);
```

46194 **DESCRIPTION**

46195 The *posix_fadvise()* function shall advise the implementation on the expected behavior of the
 46196 application with respect to the data in the file associated with the open file descriptor *fd*, starting
 46197 at *offset* and continuing for *len* bytes. The specified range need not currently exist in the file. If *len*
 46198 is zero, all data following *offset* is specified. The implementation may use this information to
 46199 optimize handling of the specified data. The *posix_fadvise()* function shall have no effect on the
 46200 semantics of other operations on the specified data, although it may affect the performance of
 46201 other operations.

46202 The advice to be applied to the data is specified by the *advice* parameter and may be one of the
 46203 following values:

46204 **POSIX_FADV_NORMAL**

46205 Specifies that the application has no advice to give on its behavior with respect to the
 46206 specified data. It is the default characteristic if no advice is given for an open file.

46207 **POSIX_FADV_SEQUENTIAL**

46208 Specifies that the application expects to access the specified data sequentially from lower
 46209 offsets to higher offsets.

46210 **POSIX_FADV_RANDOM**

46211 Specifies that the application expects to access the specified data in a random order.

46212 **POSIX_FADV_WILLNEED**

46213 Specifies that the application expects to access the specified data in the near future.

46214 **POSIX_FADV_DONTNEED**

46215 Specifies that the application expects that it will not access the specified data in the near
 46216 future.

46217 **POSIX_FADV_NOREUSE**

46218 Specifies that the application expects to access the specified data once and then not reuse it
 46219 thereafter.

46220 These values are defined in **<fcntl.h>**.

46221 **RETURN VALUE**

46222 Upon successful completion, *posix_fadvise()* shall return zero; otherwise, an error number shall
 46223 be returned to indicate the error.

46224 **ERRORS**

46225 The *posix_fadvise()* function shall fail if:

46226	[EBADF]	The <i>fd</i> argument is not a valid file descriptor.
46227	[EINVAL]	The value of <i>advice</i> is invalid, or the value of <i>len</i> is less than zero.
46228	[ESPIPE]	The <i>fd</i> argument is associated with a pipe or FIFO.

46229 EXAMPLES

46230 None.

46231 APPLICATION USAGE

46232 The *posix_fadvise()* function is part of the Advisory Information option and need not be
46233 provided on all implementations.

46234 RATIONALE

46235 None.

46236 FUTURE DIRECTIONS

46237 None.

46238 SEE ALSO

46239 *posix_madvise()*

46240 XBD **<fcntl.h>**

46241 CHANGE HISTORY

46242 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

46243 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

46244 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/68 is applied, changing the function
46245 prototype in the SYNOPSIS section. The previous prototype was not large file-aware, and the
46246 standard developers felt it acceptable to make this change before implementations of this
46247 function become widespread.

46248 Issue 7

46249 Austin Group Interpretation 1003.1-2001 #024 is applied, changing the definition of the
46250 [EINVAL] error.

posix_fallocate()

System Interfaces

46251 **NAME**46252 **posix_fallocate** — file space control (**ADVANCED REALTIME**)46253 **SYNOPSIS**

```
46254 ADV #include <fcntl.h>
46255 int posix_fallocate(int fd, off_t offset, off_t len);
```

46256 **DESCRIPTION**

46257 The *posix_fallocate()* function shall ensure that any required storage for regular file data starting
 46258 at *offset* and continuing for *len* bytes is allocated on the file system storage media. If
 46259 *posix_fallocate()* returns successfully, subsequent writes to the specified file data shall not fail due
 46260 to the lack of free space on the file system storage media.

46261 If the *offset+len* is beyond the current file size, then *posix_fallocate()* shall adjust the file size to
 46262 *offset+len*. Otherwise, the file size shall not be changed.

46263 It is implementation-defined whether a previous *posix_fadvise()* call influences allocation
 46264 strategy.

46265 Space allocated via *posix_fallocate()* shall be freed by a successful call to *creat()* or *open()* that
 46266 truncates the size of the file. Space allocated via *posix_fallocate()* may be freed by a successful call
 46267 to *ftruncate()* that reduces the file size to a size smaller than *offset+len*.

46268 **RETURN VALUE**

46269 Upon successful completion, *posix_fallocate()* shall return zero; otherwise, an error number shall
 46270 be returned to indicate the error.

46271 **ERRORS**

46272 The *posix_fallocate()* function shall fail if:

46273 [EBADF] The *fd* argument is not a valid file descriptor.

46274 [EBADF] The *fd* argument references a file that was opened without write permission.

46275 [EFBIG] The value of *offset+len* is greater than the maximum file size.

46276 [EINTR] A signal was caught during execution.

46277 [EINVAL] The *len* argument is less than zero, or the *offset* argument is less than zero, or
 46278 the underlying file system does not support this operation.

46279 [EIO] An I/O error occurred while reading from or writing to a file system.

46280 [ENODEV] The *fd* argument does not refer to a regular file.

46281 [ENOSPC] There is insufficient free space remaining on the file system storage media.

46282 [ESPIPE] The *fd* argument is associated with a pipe or FIFO.

46283 The *posix_fallocate()* function may fail if:

46284 [EINVAL] The *len* argument is zero.

EXAMPLES

None.

APPLICATION USAGE

The *posix_fallocate()* function is part of the Advisory Information option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO*creat()*, *ftruncate()*, *open()*, *unlink()*XBD **<fcntl.h>****CHANGE HISTORY**

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/69 is applied, changing the function prototype in the SYNOPSIS section. The previous prototype was not large file-aware, and the standard developers felt it acceptable to make this change before implementations of this function become widespread.

Issue 7

Austin Group Interpretations 1003.1-2001 #022, #024, and #162 are applied, changing the definition of the [EINVAL] error.

posix_madvise()

System Interfaces

46307 **NAME**

46308 **posix_madvise** — memory advisory information and alignment control (**ADVANCED**
 46309 **REALTIME**)

46310 **SYNOPSIS**

```
46311 ADV #include <sys/mman.h>
46312 int posix_madvise(void *addr, size_t len, int advice);
```

46313 **DESCRIPTION**

46314 The *posix_madvise()* function shall advise the implementation on the expected behavior of the
 46315 application with respect to the data in the memory starting at address *addr*, and continuing for
 46316 *len* bytes. The implementation may use this information to optimize handling of the specified
 46317 data. The *posix_madvise()* function shall have no effect on the semantics of access to memory in
 46318 the specified range, although it may affect the performance of access.

46319 The implementation may require that *addr* be a multiple of the page size, which is the value
 46320 returned by *sysconf()* when the name value *_SC_PAGESIZE* is used.

46321 The advice to be applied to the memory range is specified by the *advice* parameter and may be
 46322 one of the following values:

46323 **POSIX_MADV_NORMAL**

46324 Specifies that the application has no advice to give on its behavior with respect to the
 46325 specified range. It is the default characteristic if no advice is given for a range of memory.

46326 **POSIX_MADV_SEQUENTIAL**

46327 Specifies that the application expects to access the specified range sequentially from lower
 46328 addresses to higher addresses.

46329 **POSIX_MADV_RANDOM**

46330 Specifies that the application expects to access the specified range in a random order.

46331 **POSIX_MADV_WILLNEED**

46332 Specifies that the application expects to access the specified range in the near future.

46333 **POSIX_MADV_DONTNEED**

46334 Specifies that the application expects that it will not access the specified range in the near
 46335 future.

46336 These values are defined in the **<sys/mman.h>** header.

46337 **RETURN VALUE**

46338 Upon successful completion, *posix_madvise()* shall return zero; otherwise, an error number shall
 46339 be returned to indicate the error.

46340 **ERRORS**

46341 The *posix_madvise()* function shall fail if:

46342 [EINVAL] The value of *advice* is invalid.

46343 [ENOMEM] Addresses in the range starting at *addr* and continuing for *len* bytes are partly
 46344 or completely outside the range allowed for the address space of the calling
 46345 process.

System Interfaces

posix_madvise()

46346 The *posix_madvise()* function may fail if:

46347 [EINVAL] The value of *addr* is not a multiple of the value returned by *sysconf()* when the
46348 name value *_SC_PAGESIZE* is used.

46349 [EINVAL] The value of *len* is zero.

EXAMPLES

46351 None.

APPLICATION USAGE

46353 The *posix_madvise()* function is part of the Advisory Information option and need not be
46354 provided on all implementations.

RATIONALE

46356 None.

FUTURE DIRECTIONS

46358 None.

SEE ALSO

46360 *mmap()*, *posix_fadvise()*, *sysconf()*

46361 XBD <sys/mman.h>

CHANGE HISTORY

46363 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

46364 IEEE PASC Interpretation 1003.1 #102 is applied.

posix_mem_offset()

System Interfaces

NAME

posix_mem_offset — find offset and length of a mapped typed memory block (ADVANCED REALTIME)

SYNOPSIS

```
#include <sys/mman.h>

int posix_mem_offset(const void *restrict addr, size_t len,
                    off_t *restrict off, size_t *restrict contig_len,
                    int *restrict fildes);
```

DESCRIPTION

The *posix_mem_offset()* function shall return in the variable pointed to by *off* a value that identifies the offset (or location), within a memory object, of the memory block currently mapped at *addr*. The function shall return in the variable pointed to by *fildes*, the descriptor used (via *mmap()*) to establish the mapping which contains *addr*. If that descriptor was closed since the mapping was established, the returned value of *fildes* shall be *-1*. The *len* argument specifies the length of the block of the memory object the user wishes the offset for; upon return, the value pointed to by *contig_len* shall equal either *len*, or the length of the largest contiguous block of the memory object that is currently mapped to the calling process starting at *addr*, whichever is smaller.

If the memory object mapped at *addr* is a typed memory object, then if the *off* and *contig_len* values obtained by calling *posix_mem_offset()* are used in a call to *mmap()* with a file descriptor that refers to the same memory pool as *fildes* (either through the same port or through a different port), and that was opened with neither the *POSIX_TYPED_MEM_ALLOCATE* nor the *POSIX_TYPED_MEM_ALLOCATE_CONTIG* flag, the typed memory area that is mapped shall be exactly the same area that was mapped at *addr* in the address space of the process that called *posix_mem_offset()*.

If the memory object specified by *fildes* is not a typed memory object, then the behavior of this function is implementation-defined.

RETURN VALUE

Upon successful completion, the *posix_mem_offset()* function shall return zero; otherwise, the corresponding error status value shall be returned.

ERRORS

The *posix_mem_offset()* function shall fail if:

[EACCES] The process has not mapped a memory object supported by this function at the given address *addr*.

This function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

46406 **FUTURE DIRECTIONS**

46407 None.

46408 **SEE ALSO**46409 *mmap()*, *posix_typed_mem_open()*46410 XBD [<sys/mman.h>](#)46411 **CHANGE HISTORY**

46412 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_memalign()

System Interfaces

46413 **NAME**46414 **posix_memalign** — aligned memory allocation (**ADVANCED REALTIME**)46415 **SYNOPSIS**

```
46416 ADV #include <stdlib.h>
46417 int posix_memalign(void **memptr, size_t alignment, size_t size);
```

46418 **DESCRIPTION**

46419 The *posix_memalign()* function shall allocate *size* bytes aligned on a boundary specified by
 46420 *alignment*, and shall return a pointer to the allocated memory in *memptr*. The value of *alignment*
 46421 shall be a power of two multiple of *sizeof(void *)*.

46422 Upon successful completion, the value pointed to by *memptr* shall be a multiple of *alignment*.

46423 If the size of the space requested is 0, the behavior is implementation-defined; the value returned
 46424 in *memptr* shall be either a null pointer or a unique pointer.

46425 CX The *free()* function shall deallocate memory that has previously been allocated by
 46426 *posix_memalign()*.

46427 **RETURN VALUE**

46428 Upon successful completion, *posix_memalign()* shall return zero; otherwise, an error number
 46429 shall be returned to indicate the error.

46430 **ERRORS**

46431 The *posix_memalign()* function shall fail if:

46432	[EINVAL]	The value of the alignment parameter is not a power of two multiple of <i>sizeof(void *)</i> .
46433		
46434	[ENOMEM]	There is insufficient memory available with the requested alignment.

46435 **EXAMPLES**

46436 None.

46437 **APPLICATION USAGE**

46438 The *posix_memalign()* function is part of the Advisory Information option and need not be
 46439 provided on all implementations.

46440 **RATIONALE**

46441 None.

46442 **FUTURE DIRECTIONS**

46443 None.

46444 **SEE ALSO**

46445 *free()*, *malloc()*
 46446 XBD *<stdlib.h>*

46447 **CHANGE HISTORY**

46448 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

46449 In the SYNOPSIS, the inclusion of *<sys/types.h>* is no longer required.

46450 **Issue 7**

46451 Austin Group Interpretation 1003.1-2001 #058 is applied, clarifying the value of the *alignment*
 46452 argument in the DESCRIPTION.

46453
46454

Austin Group Interpretation 1003.1-2001 #152 is applied, clarifying the behavior when the size of the space requested is 0.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_openpt()

System Interfaces

46455 **NAME**46456 **posix_openpt** — open a pseudo-terminal device46457 **SYNOPSIS**

```

46458 XSI      #include <stdlib.h>
46459          #include <fcntl.h>
46460          int posix_openpt(int oflag);

```

46461 **DESCRIPTION**

46462 The *posix_openpt()* function shall establish a connection between a master device for a pseudo-terminal and a file descriptor. The file descriptor is used by other I/O functions that refer to that pseudo-terminal.

46465 The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

46467 Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in *<fcntl.h>*:

46469 **O_RDWR** Open for reading and writing.

46470 **O_NOCTTY** If set *posix_openpt()* shall not cause the terminal device to become the controlling terminal for the process.

46472 The behavior of other values for the *oflag* argument is unspecified.

46473 **RETURN VALUE**

46474 Upon successful completion, the *posix_openpt()* function shall open a master pseudo-terminal device and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, -1 shall be returned and *errno* set to indicate the error.

46477 **ERRORS**

46478 The *posix_openpt()* function shall fail if:

46479 **[EMFILE]** All file descriptors available to the process are currently open.

46480 **[ENFILE]** The maximum allowable number of files is currently open in the system.

46481 The *posix_openpt()* function may fail if:

46482 **[EINVAL]** The value of *oflag* is not valid.

46483 **[EAGAIN]** Out of pseudo-terminal resources.

46484 **[ENOSR]** Out of STREAMS resources.

46485 **EXAMPLES**

46486 Opening a Pseudo-Terminal and Returning the Name of the Slave Device and a File Descriptor

```

46488 #include <fcntl.h>
46489 #include <stdio.h>
46490
46491 int masterfd, slavefd;
46492 char *slavedevice;
46493
46494 masterfd = posix_openpt (O_RDWR|O_NOCTTY);
46495
46496 if (masterfd == -1
46497     || grantpt (masterfd) == -1

```



```

46495         || unlockpt (masterfd) == -1
46496         || (slavedevice = ptsname (masterfd)) == NULL)
46497         return -1;

46498     printf("slave device is: %s\n", slavedevice);

46499     slavefd = open(slavedevice, O_RDWR|O_NOCTTY);
46500     if (slavefd < 0)
46501         return -1;

```

APPLICATION USAGE

46503 This function is a method for portably obtaining a file descriptor of a master terminal device for
 46504 a pseudo-terminal. The *grantpt()* and *ptsname()* functions can be used to manipulate mode and
 46505 ownership permissions, and to obtain the name of the slave device, respectively.

RATIONALE

46507 The standard developers considered the matter of adding a special device for cloning master
 46508 pseudo-terminals: the */dev/ptmx* device. However, consensus could not be reached, and it was
 46509 felt that adding a new function would permit other implementations. The *posix_openpt()*
 46510 function is designed to complement the *grantpt()*, *ptsname()*, and *unlockpt()* functions.

46511 On implementations supporting the */dev/ptmx* clone device, opening the master device of a
 46512 pseudo-terminal is simply:

```

46513 mfdp = open("/dev/ptmx", oflag );
46514 if (mfdp < 0)
46515     return -1;

```

FUTURE DIRECTIONS

46517 None.

SEE ALSO

46519 *grantpt()*, *open()*, *ptsname()*, *unlockpt()*

46520 XBD *<fcntl.h>*, *<stdlib.h>*

CHANGE HISTORY

46522 First released in Issue 6.

Issue 7

46524 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

46525 SD5-XSH-ERN-51 is applied, correcting an error in the EXAMPLES section.

posix_spawn()

System Interfaces

NAMEposix_spawn, posix_spawnnp — spawn a process (**ADVANCED REALTIME**)**SYNOPSIS**

```

#include <spawn.h>

int posix_spawn(pid_t *restrict pid, const char *restrict path,
    const posix_spawn_file_actions_t *file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char *const envp[restrict]);
int posix_spawnnp(pid_t *restrict pid, const char *restrict file,
    const posix_spawn_file_actions_t *file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char *const envp[restrict]);

```

DESCRIPTION

The *posix_spawn()* and *posix_spawnnp()* functions shall create a new process (child process) from the specified process image. The new process image shall be constructed from a regular executable file called the new process image file.

When a C program is executed as the result of this call, it shall be entered as a C-language function call as follows:

```
int main(int argc, char *argv[]);
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. In addition, the following variable:

```
extern char **environ;
```

shall be initialized as a pointer to an array of character pointers to the environment strings.

The argument *argv* is an array of character pointers to null-terminated strings. The last member of this array shall be a null pointer and is not counted in *argc*. These strings constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename that is associated with the process image being started by the *posix_spawn()* or *posix_spawnnp()* function.

The argument *envp* is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The environment array is terminated by a null pointer.

The number of bytes available for the combined argument and environment lists of the child process is {ARG_MAX}. The implementation shall specify in the system documentation (see XBD Chapter 2, on page 15) whether any list overhead, such as length words, null terminators, pointers, or alignment bytes, is included in this total.

The *path* argument to *posix_spawn()* is a pathname that identifies the new process image file to execute.

The *file* parameter to *posix_spawnnp()* shall be used to construct a pathname that identifies the new process image file. If the *file* parameter contains a <slash> character, the *file* parameter shall be used as the pathname for the new process image file. Otherwise, the path prefix for this file shall be obtained by a search of the directories passed as the environment variable *PATH* (see XBD Chapter 8, on page 173). If this environment variable is not defined, the results of the search are implementation-defined.

If *file_actions* is a null pointer, then file descriptors open in the calling process shall remain open

in the child process, except for those whose close-on-exec flag `FD_CLOEXEC` is set (see *fcntl()*). For those file descriptors that remain open, all attributes of the corresponding open file descriptions, including file locks (see *fcntl()*), shall remain unchanged.

If *file_actions* is not NULL, then the file descriptors open in the child process shall be those open in the calling process as modified by the spawn file actions object pointed to by *file_actions* and the `FD_CLOEXEC` flag of each remaining open file descriptor after the spawn file actions have been processed. The effective order of processing the spawn file actions shall be:

1. The set of open file descriptors for the child process shall initially be the same set as is open for the calling process. All attributes of the corresponding open file descriptions, including file locks (see *fcntl()*), shall remain unchanged.
2. The signal mask, signal default actions, and the effective user and group IDs for the child process shall be changed as specified in the attributes object referenced by *attrp*.
3. The file actions specified by the spawn file actions object shall be performed in the order in which they were added to the spawn file actions object.
4. Any file descriptor that has its `FD_CLOEXEC` flag set (see *fcntl()*) shall be closed.

The **posix_spawnattr_t** spawn attributes object type is defined in **<spawn.h>**. It shall contain at least the attributes defined below.

If the `POSIX_SPAWN_SETPGROUP` flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is non-zero, then the child's process group shall be as specified in the *spawn-pgroup* attribute of the object referenced by *attrp*.

As a special case, if the `POSIX_SPAWN_SETPGROUP` flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is set to zero, then the child shall be in a new process group with a process group ID equal to its process ID.

If the `POSIX_SPAWN_SETPGROUP` flag is not set in the *spawn-flags* attribute of the object referenced by *attrp*, the new child process shall inherit the parent's process group.

If the `POSIX_SPAWN_SETSCHEDPARAM` flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, but `POSIX_SPAWN_SETSCHEDULER` is not set, the new process image shall initially have the scheduling policy of the calling process with the scheduling parameters specified in the *spawn-schedparam* attribute of the object referenced by *attrp*.

If the `POSIX_SPAWN_SETSCHEDULER` flag is set in the *spawn-flags* attribute of the object referenced by *attrp* (regardless of the setting of the `POSIX_SPAWN_SETSCHEDPARAM` flag), the new process image shall initially have the scheduling policy specified in the *spawn-schedpolicy* attribute of the object referenced by *attrp* and the scheduling parameters specified in the *spawn-schedparam* attribute of the same object.

The `POSIX_SPAWN_RESETPIDS` flag in the *spawn-flags* attribute of the object referenced by *attrp* governs the effective user ID of the child process. If this flag is not set, the child process shall inherit the effective user ID of the parent process. If this flag is set, the effective user ID of the child process shall be reset to the parent's real user ID. In either case, if the set-user-ID mode bit of the new process image file is set, the effective user ID of the child process shall become that file's owner ID before the new process image begins execution.

The `POSIX_SPAWN_RESETPIDS` flag in the *spawn-flags* attribute of the object referenced by *attrp* also governs the effective group ID of the child process. If this flag is not set, the child process shall inherit the effective group ID of the parent process. If this flag is set, the effective group ID of the child process shall be reset to the parent's real group ID. In either case, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the child process shall

posix_spawn()*System Interfaces*

become that file's group ID before the new process image begins execution.

If the POSIX_SPAWN_SETSIGMASK flag is set in the *spawn_flags* attribute of the object referenced by *attrp*, the child process shall initially have the signal mask specified in the *spawn_sigmask* attribute of the object referenced by *attrp*.

If the POSIX_SPAWN_SETSIGDEF flag is set in the *spawn_flags* attribute of the object referenced by *attrp*, the signals specified in the *spawn_sigdefault* attribute of the same object shall be set to their default actions in the child process. Signals set to the default action in the parent process shall be set to the default action in the child process.

Signals set to be caught by the calling process shall be set to the default action in the child process.

Except for SIGCHLD, signals set to be ignored by the calling process image shall be set to be ignored by the child process, unless otherwise specified by the POSIX_SPAWN_SETSIGDEF flag being set in the *spawn_flags* attribute of the object referenced by *attrp* and the signals being indicated in the *spawn_sigdefault* attribute of the object referenced by *attrp*.

If the SIGCHLD signal is set to be ignored by the calling process, it is unspecified whether the SIGCHLD signal is set to be ignored or to the default action in the child process, unless otherwise specified by the POSIX_SPAWN_SETSIGDEF flag being set in the *spawn_flags* attribute of the object referenced by *attrp* and the SIGCHLD signal being indicated in the *spawn_sigdefault* attribute of the object referenced by *attrp*.

If the value of the *attrp* pointer is NULL, then the default values are used.

All process attributes, other than those influenced by the attributes set in the object referenced by *attrp* as specified above or by the file descriptor manipulations specified in *file_actions*, shall appear in the new process image as though *fork()* had been called to create a child process and then a member of the *exec* family of functions had been called by the child process to execute the new process image.

It is implementation-defined whether the fork handlers are run when *posix_spawn()* or *posix_spawnnp()* is called.

RETURN VALUE

Upon successful completion, *posix_spawn()* and *posix_spawnnp()* shall return the process ID of the child process to the parent process, in the variable pointed to by a non-NULL *pid* argument, and shall return zero as the function return value. Otherwise, no child process shall be created, the value stored in the variable pointed to by a non-NULL *pid* is unspecified, and an error number shall be returned as the function return value to indicate the error. If the *pid* argument is a null pointer, the process ID of the child is not returned to the caller.

ERRORS

These functions may fail if:

[EINVAL] The value specified by *file_actions* or *attrp* is invalid.

If this error occurs after the calling process successfully returns from the *posix_spawn()* or *posix_spawnnp()* function, the child process may exit with exit status 127.

If *posix_spawn()* or *posix_spawnnp()* fail for any of the reasons that would cause *fork()* or one of the *exec* family of functions to fail, an error value shall be returned as described by *fork()* and *exec*, respectively (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127).

If POSIX_SPAWN_SETPGROUP is set in the *spawn_flags* attribute of the object referenced by *attrp*, and *posix_spawn()* or *posix_spawnnp()* fails while changing the child's process group, an

error value shall be returned as described by *setpgid()* (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127).

PS If *POSIX_SPAWN_SETSCHEDPARAM* is set and *POSIX_SPAWN_SETSCHEDULER* is not set in the *spawn_flags* attribute of the object referenced by *attrp*, then if *posix_spawn()* or *posix_spawnnp()* fails for any of the reasons that would cause *sched_setparam()* to fail, an error value shall be returned as described by *sched_setparam()* (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127).

If *POSIX_SPAWN_SETSCHEDULER* is set in the *spawn_flags* attribute of the object referenced by *attrp*, and if *posix_spawn()* or *posix_spawnnp()* fails for any of the reasons that would cause *sched_setscheduler()* to fail, an error value shall be returned as described by *sched_setscheduler()* (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127).

If the *file_actions* argument is not NULL, and specifies any *close*, *dup2*, or *open* actions to be performed, and if *posix_spawn()* or *posix_spawnnp()* fails for any of the reasons that would cause *close()*, *dup2()*, or *open()* to fail, an error value shall be returned as described by *close()*, *dup2()*, and *open()*, respectively (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127). An open file action may, by itself, result in any of the errors described by *close()* or *dup2()*, in addition to those described by *open()*.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

The *posix_spawn()* function and its close relation *posix_spawnnp()* have been introduced to overcome the following perceived difficulties with *fork()*: the *fork()* function is difficult or impossible to implement without swapping or dynamic address translation.

- Swapping is generally too slow for a realtime environment.
- Dynamic address translation is not available everywhere that POSIX might be useful.
- Processes are too useful to simply option out of POSIX whenever it must run without address translation or other MMU services.

Thus, POSIX needs process creation and file execution primitives that can be efficiently implemented without address translation or other MMU services.

The *posix_spawn()* function is implementable as a library routine, but both *posix_spawn()* and *posix_spawnnp()* are designed as kernel operations. Also, although they may be an efficient replacement for many *fork()/exec* pairs, their goal is to provide useful process creation primitives for systems that have difficulty with *fork()*, not to provide drop-in replacements for *fork()/exec*.

This view of the role of *posix_spawn()* and *posix_spawnnp()* influenced the design of their API. It does not attempt to provide the full functionality of *fork()/exec* in which arbitrary user-specified operations of any sort are permitted between the creation of the child process and the execution of the new process image; any attempt to reach that level would need to provide a programming language as parameters. Instead, *posix_spawn()* and *posix_spawnnp()* are process creation primitives like the *Start_Process* and *Start_Process_Search* Ada language bindings package *POSIX_Process_Primitives* and also like those in many operating systems that are not UNIX systems, but with some POSIX-specific additions.

posix_spawn()*System Interfaces*

To achieve its coverage goals, *posix_spawn()* and *posix_spawnnp()* have control of six types of inheritance: file descriptors, process group ID, user and group ID, signal mask, scheduling, and whether each signal ignored in the parent will remain ignored in the child, or be reset to its default action in the child.

Control of file descriptors is required to allow an independently written child process image to access data streams opened by and even generated or read by the parent process without being specifically coded to know which parent files and file descriptors are to be used. Control of the process group ID is required to control how the job control of the child process relates to that of the parent.

Control of the signal mask and signal defaulting is sufficient to support the implementation of *system()*. Although support for *system()* is not explicitly one of the goals for *posix_spawn()* and *posix_spawnnp()*, it is covered under the “at least 50%” coverage goal.

The intention is that the normal file descriptor inheritance across *fork()*, the subsequent effect of the specified spawn file actions, and the normal file descriptor inheritance across one of the *exec* family of functions should fully specify open file inheritance. The implementation need make no decisions regarding the set of open file descriptors when the child process image begins execution, those decisions having already been made by the caller and expressed as the set of open file descriptors and their FD_CLOEXEC flags at the time of the call and the spawn file actions object specified in the call. We have been assured that in cases where the POSIX *Start_Process* Ada primitives have been implemented in a library, this method of controlling file descriptor inheritance may be implemented very easily.

We can identify several problems with *posix_spawn()* and *posix_spawnnp()*, but there does not appear to be a solution that introduces fewer problems. Environment modification for child process attributes not specifiable via the *attrp* or *file_actions* arguments must be done in the parent process, and since the parent generally wants to save its context, it is more costly than similar functionality with *fork()/exec*. It is also complicated to modify the environment of a multi-threaded process temporarily, since all threads must agree when it is safe for the environment to be changed. However, this cost is only borne by those invocations of *posix_spawn()* and *posix_spawnnp()* that use the additional functionality. Since extensive modifications are not the usual case, and are particularly unlikely in time-critical code, keeping much of the environment control out of *posix_spawn()* and *posix_spawnnp()* is appropriate design.

The *posix_spawn()* and *posix_spawnnp()* functions do not have all the power of *fork()/exec*. This is to be expected. The *fork()* function is a wonderfully powerful operation. We do not expect to duplicate its functionality in a simple, fast function with no special hardware requirements. It is worth noting that *posix_spawn()* and *posix_spawnnp()* are very similar to the process creation operations on many operating systems that are not UNIX systems.

Requirements

The requirements for *posix_spawn()* and *posix_spawnnp()* are:

- They must be implementable without an MMU or unusual hardware.
- They must be compatible with existing POSIX standards.

Additional goals are:

- They should be efficiently implementable.
- They should be able to replace at least 50% of typical executions of *fork()*.

- A system with *posix_spawn()* and *posix_spawnnp()* and without *fork()* should be useful, at least for realtime applications.
- A system with *fork()* and the *exec* family should be able to implement *posix_spawn()* and *posix_spawnnp()* as library routines.

Two-Syntax

POSIX *exec* has several calling sequences with approximately the same functionality. These appear to be required for compatibility with existing practice. Since the existing practice for the *posix_spawn*()* functions is otherwise substantially unlike POSIX, we feel that simplicity outweighs compatibility. There are, therefore, only two names for the *posix_spawn*()* functions.

The parameter list does not differ between *posix_spawn()* and *posix_spawnnp()*; *posix_spawnnp()* interprets the second parameter more elaborately than *posix_spawn()*.

Compatibility with POSIX.5 (Ada)

The *Start_Process* and *Start_Process_Search* procedures from the *POSIX_Process_Primitives* package from the Ada language binding to POSIX.1 encapsulate *fork()* and *exec* functionality in a manner similar to that of *posix_spawn()* and *posix_spawnnp()*. Originally, in keeping with our simplicity goal, the standard developers had limited the capabilities of *posix_spawn()* and *posix_spawnnp()* to a subset of the capabilities of *Start_Process* and *Start_Process_Search*; certain non-default capabilities were not supported. However based on suggestions by the ballot group to improve file descriptor mapping or drop it, and on the advice of an Ada Language Bindings working group member, the standard developers decided that *posix_spawn()* and *posix_spawnnp()* should be sufficiently powerful to implement *Start_Process* and *Start_Process_Search*. The rationale is that if the Ada language binding to such a primitive had already been approved as an IEEE standard, there can be little justification for not approving the functionally-equivalent parts of a C binding. The only three capabilities provided by *posix_spawn()* and *posix_spawnnp()* that are not provided by *Start_Process* and *Start_Process_Search* are optionally specifying the child's process group ID, the set of signals to be reset to default signal handling in the child process, and the child's scheduling policy and parameters.

For the Ada language binding for *Start_Process* to be implemented with *posix_spawn()*, that binding would need to explicitly pass an empty signal mask and the parent's environment to *posix_spawn()* whenever the caller of *Start_Process* allowed these arguments to default, since *posix_spawn()* does not provide such defaults. The ability of *Start_Process* to mask user-specified signals during its execution is functionally unique to the Ada language binding and must be dealt with in the binding separately from the call to *posix_spawn()*.

Process Group

The process group inheritance field can be used to join the child process with an existing process group. By assigning a value of zero to the *spawn-pgroup* attribute of the object referenced by *attrp*, the *setpgid()* mechanism will place the child process in a new process group.

posix_spawn()*System Interfaces***Threads**

Without the *posix_spawn()* and *posix_spawnnp()* functions, systems without address translation can still use threads to give an abstraction of concurrency. In many cases, thread creation suffices, but it is not always a good substitute. The *posix_spawn()* and *posix_spawnnp()* functions are considerably “heavier” than thread creation. Processes have several important attributes that threads do not. Even without address translation, a process may have base-and-bound memory protection. Each process has a process environment including security attributes and file capabilities, and powerful scheduling attributes. Processes abstract the behavior of non-uniform-memory-architecture multi-processors better than threads, and they are more convenient to use for activities that are not closely linked.

The *posix_spawn()* and *posix_spawnnp()* functions may not bring support for multiple processes to every configuration. Process creation is not the only piece of operating system support required to support multiple processes. The total cost of support for multiple processes may be quite high in some circumstances. Existing practice shows that support for multiple processes is uncommon and threads are common among “tiny kernels”. There should, therefore, probably continue to be AEPs for operating systems with only one process.

Asynchronous Error Notification

A library implementation of *posix_spawn()* or *posix_spawnnp()* may not be able to detect all possible errors before it forks the child process. POSIX.1-2008 provides for an error indication returned from a child process which could not successfully complete the spawn operation via a special exit status which may be detected using the status value returned by *wait()*, *waitid()*, and *waitpid()*.

The *stat_val* interface and the macros used to interpret it are not well suited to the purpose of returning API errors, but they are the only path available to a library implementation. Thus, an implementation may cause the child process to exit with exit status 127 for any error detected during the spawn process after the *posix_spawn()* or *posix_spawnnp()* function has successfully returned.

The standard developers had proposed using two additional macros to interpret *stat_val*. The first, *WIFSPAWNFAIL*, would have detected a status that indicated that the child exited because of an error detected during the *posix_spawn()* or *posix_spawnnp()* operations rather than during actual execution of the child process image; the second, *WSPAWNERRNO*, would have extracted the error value if *WIFSPAWNFAIL* indicated a failure. Unfortunately, the ballot group strongly opposed this because it would make a library implementation of *posix_spawn()* or *posix_spawnnp()* dependent on kernel modifications to *waitpid()* to be able to embed special information in *stat_val* to indicate a spawn failure.

The 8 bits of child process exit status that are guaranteed by POSIX.1-2008 to be accessible to the waiting parent process are insufficient to disambiguate a spawn error from any other kind of error that may be returned by an arbitrary process image. No other bits of the exit status are required to be visible in *stat_val*, so these macros could not be strictly implemented at the library level. Reserving an exit status of 127 for such spawn errors is consistent with the use of this value by *system()* and *popen()* to signal failures in these operations that occur after the function has returned but before a shell is able to execute. The exit status of 127 does not uniquely identify this class of error, nor does it provide any detailed information on the nature of the failure. Note that a kernel implementation of *posix_spawn()* or *posix_spawnnp()* is permitted (and encouraged) to return any possible error as the function value, thus providing more detailed failure information to the parent process.

Thus, no special macros are available to isolate asynchronous *posix_spawn()* or *posix_spawnnp()*

errors. Instead, errors detected by the *posix_spawn()* or *posix_spawnnp()* operations in the context of the child process before the new process image executes are reported by setting the child's exit status to 127. The calling process may use the WIFEXITED and WEXITSTATUS macros on the *stat_val* stored by the *wait()* or *waitpid()* functions to detect spawn failures to the extent that other status values with which the child process image may exit (before the parent can conclusively determine that the child process image has begun execution) are distinct from exit status 127.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm(), *chmod()*, *close()*, *dup()*, *exec*, *exit()*, *fcntl()*, *fork()*, *fstatat()*, *kill()*, *open()*, *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*, *posix_spawn_file_actions_destroy()*, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_getsigmask()*, *sched_setparam()*, *sched_setscheduler()*, *setpgid()*, *setuid()*, *times()*, *wait()*, *waitid()*

XBD Chapter 8 (on page 173), [**<spawn.h>**](#)

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #103 is applied, noting that the signal default actions are changed as well as the signal mask in step 2.

IEEE PASC Interpretation 1003.1 #132 is applied.

Issue 7

Functionality relating to the Threads option is moved to the Base.

posix_spawn_file_actions_addclose()

System Interfaces

NAME

posix_spawn_file_actions_addclose, posix_spawn_file_actions_addopen — add close or open action to spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>

int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t
    *file_actions, int fildes);
int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t
    *restrict file_actions, int fildes,
    const char *restrict path, int oflag, mode_t mode);
```

DESCRIPTION

These functions shall add or delete a close or open action to a spawn file actions object.

A spawn file actions object is of type **posix_spawn_file_actions_t** (defined in **<spawn.h>**) and is used to specify a series of actions to be performed by a *posix_spawn()* or *posix_spawnnp()* operation in order to arrive at the set of open file descriptors for the child process given the set of open file descriptors of the parent. POSIX.1-2008 does not define comparison or assignment operators for the type **posix_spawn_file_actions_t**.

A spawn file actions object, when passed to *posix_spawn()* or *posix_spawnnp()*, shall specify how the set of open file descriptors in the calling process is transformed into a set of potentially open file descriptors for the spawned process. This transformation shall be as if the specified sequence of actions was performed exactly once, in the context of the spawned process (prior to execution of the new process image), in the order in which the actions were added to the object; additionally, when the new process image is executed, any file descriptor (from this new set) which has its FD_CLOEXEC flag set shall be closed (see *posix_spawn()*).

The *posix_spawn_file_actions_addclose()* function shall add a *close* action to the object referenced by *file_actions* that shall cause the file descriptor *fildes* to be closed (as if *close(fildes)* had been called) when a new process is spawned using this file actions object.

The *posix_spawn_file_actions_addopen()* function shall add an *open* action to the object referenced by *file_actions* that shall cause the file named by *path* to be opened (as if *open(path, oflag, mode)* had been called, and the returned file descriptor, if not *fildes*, had been changed to *fildes*) when a new process is spawned using this file actions object. If *fildes* was already an open file descriptor, it shall be closed before the new file is opened.

The string described by *path* shall be copied by the *posix_spawn_file_actions_addopen()* function.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall fail if:

[EBADF] The value specified by *fildes* is negative or greater than or equal to {OPEN_MAX}.

These functions may fail if:

[EINVAL] The value specified by *file_actions* is invalid.

[ENOMEM] Insufficient memory exists to add to the spawn file actions object.

It shall not be considered an error for the *fildev* argument passed to these functions to specify a file descriptor for which the specified operation could not be performed at the time of the call. Any such error will be detected when the associated file actions object is later used during a *posix_spawn()* or *posix_spawnnp()* operation.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

A spawn file actions object may be initialized to contain an ordered sequence of *close()*, *dup2()*, and *open()* operations to be used by *posix_spawn()* or *posix_spawnnp()* to arrive at the set of open file descriptors inherited by the spawned process from the set of open file descriptors in the parent at the time of the *posix_spawn()* or *posix_spawnnp()* call. It had been suggested that the *close()* and *dup2()* operations alone are sufficient to rearrange file descriptors, and that files which need to be opened for use by the spawned process can be handled either by having the calling process open them before the *posix_spawn()* or *posix_spawnnp()* call (and close them after), or by passing filenames to the spawned process (in *argv*) so that it may open them itself. The standard developers recommend that applications use one of these two methods when practical, since detailed error status on a failed open operation is always available to the application this way. However, the standard developers feel that allowing a spawn file actions object to specify open operations is still appropriate because:

1. It is consistent with equivalent POSIX.5 (Ada) functionality.
2. It supports the I/O redirection paradigm commonly employed by POSIX programs designed to be invoked from a shell. When such a program is the child process, it may not be designed to open files on its own.
3. It allows file opens that might otherwise fail or violate file ownership/access rights if executed by the parent process.

Regarding 2. above, note that the spawn open file action provides to *posix_spawn()* and *posix_spawnnp()* the same capability that the shell redirection operators provide to *system()*, only without the intervening execution of a shell; for example:

```
system ("myprog <file1 3<file2");
```

Regarding 3. above, note that if the calling process needs to open one or more files for access by the spawned process, but has insufficient spare file descriptors, then the open action is necessary to allow the *open()* to occur in the context of the child process after other file descriptors have been closed (that must remain open in the parent).

Additionally, if a parent is executed from a file having a "set-user-id" mode bit set and the POSIX_SPAWN_RESETIDS flag is set in the spawn attributes, a file created within the parent process will (possibly incorrectly) have the parent's effective user ID as its owner, whereas a file created via an *open()* action during *posix_spawn()* or *posix_spawnnp()* will have the parent's real ID as its owner; and an open by the parent process may successfully open a file to which the real user should not have access or fail to open a file to which the real user should have access.

posix_spawn_file_actions_addclose()

System Interfaces

File Descriptor Mapping

The standard developers had originally proposed using an array which specified the mapping of child file descriptors back to those of the parent. It was pointed out by the ballot group that it is not possible to reshuffle file descriptors arbitrarily in a library implementation of *posix_spawn()* or *posix_spawnnp()* without provision for one or more spare file descriptor entries (which simply may not be available). Such an array requires that an implementation develop a complex strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor at the wrong time.

It was noted by a member of the Ada Language Bindings working group that the approved Ada Language *Start_Process* family of POSIX process primitives use a caller-specified set of file actions to alter the normal *fork()/exec* semantics for inheritance of file descriptors in a very flexible way, yet no such problems exist because the burden of determining how to achieve the final file descriptor mapping is completely on the application. Furthermore, although the file actions interface appears frightening at first glance, it is actually quite simple to implement in either a library or the kernel.

FUTURE DIRECTIONS

None.

SEE ALSO

close(), *dup()*, *open()*, *posix_spawn()*, *posix_spawn_file_actions_adddup2()*,
posix_spawn_file_actions_destroy()

XBD <spawn.h>

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #105 is applied, adding a note to the DESCRIPTION that the string pointed to by *path* is copied by the *posix_spawn_file_actions_addopen()* function.

System Interfaces

posix_spawn_file_actions_adddup2()**NAME**

posix_spawn_file_actions_adddup2 — add dup2 action to spawn file actions object (ADVANCED REALTIME)

SYNOPSIS

```
#include <spawn.h>

int posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t
    *file_actions, int fildes, int newfildes);
```

DESCRIPTION

The *posix_spawn_file_actions_adddup2()* function shall add a *dup2()* action to the object referenced by *file_actions* that shall cause the file descriptor *fildes* to be duplicated as *newfildes* (as if *dup2(fildes, newfildes)* had been called) when a new process is spawned using this file actions object.

A spawn file actions object is as defined in *posix_spawn_file_actions_addclose()*.

RETURN VALUE

Upon successful completion, the *posix_spawn_file_actions_adddup2()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_spawn_file_actions_adddup2()* function shall fail if:

- [EBADF] The value specified by *fildes* or *newfildes* is negative or greater than or equal to {OPEN_MAX}.
- [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

The *posix_spawn_file_actions_adddup2()* function may fail if:

- [EINVAL] The value specified by *file_actions* is invalid.

It shall not be considered an error for the *fildes* argument passed to the *posix_spawn_file_actions_adddup2()* function to specify a file descriptor for which the specified operation could not be performed at the time of the call. Any such error will be detected when the associated file actions object is later used during a *posix_spawn()* or *posix_spawnnp()* operation.

EXAMPLES

None.

APPLICATION USAGE

The *posix_spawn_file_actions_adddup2()* function is part of the Spawn option and need not be provided on all implementations.

RATIONALE

Refer to the RATIONALE in *posix_spawn_file_actions_addclose()*.

FUTURE DIRECTIONS

None.

SEE ALSO

dup(), *posix_spawn()*, *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_destroy()*

XBD **<spawn.h>**

posix_spawn_file_actions_adddup2()*System Interfaces***CHANGE HISTORY**

47005	First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
47006	
47007	IEEE PASC Interpretation 1003.1 #104 is applied, noting that the [EBADF] error can apply to the
47008	<i>newfildes</i> argument in addition to <i>fildes</i> .

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***posix_spawn_file_actions_addopen()**47009 **NAME**

47010 `posix_spawn_file_actions_addopen` — add open action to spawn file actions object
47011 (**ADVANCED REALTIME**)

47012 **SYNOPSIS**

```
47013 SPN        #include <spawn.h>  
47014        int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t  
47015                *restrict file_actions, int fildes,  
47016                const char *restrict path, int oflag, mode_t mode);
```

47017 **DESCRIPTION**

47018 Refer to [*posix_spawn_file_actions_addclose\(\)*](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_spawn_file_actions_destroy()

System Interfaces

47019 **NAME**

47020 `posix_spawn_file_actions_destroy`, `posix_spawn_file_actions_init` — destroy and initialize
 47021 spawn file actions object (**ADVANCED REALTIME**)

47022 **SYNOPSIS**

```
47023 SPN      #include <spawn.h>

47024      int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t
47025          *file_actions);
47026      int posix_spawn_file_actions_init(posix_spawn_file_actions_t
47027          *file_actions);
```

47028 **DESCRIPTION**

47029 The `posix_spawn_file_actions_destroy()` function shall destroy the object referenced by `file_actions`;
 47030 the object becomes, in effect, uninitialized. An implementation may cause
 47031 `posix_spawn_file_actions_destroy()` to set the object referenced by `file_actions` to an invalid value. A
 47032 destroyed spawn file actions object can be reinitialized using `posix_spawn_file_actions_init()`; the
 47033 results of otherwise referencing the object after it has been destroyed are undefined.

47034 The `posix_spawn_file_actions_init()` function shall initialize the object referenced by `file_actions` to
 47035 contain no file actions for `posix_spawn()` or `posix_spawnnp()` to perform.

47036 A spawn file actions object is as defined in `posix_spawn_file_actions_addclose()`.

47037 The effect of initializing an already initialized spawn file actions object is undefined.

47038 **RETURN VALUE**

47039 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 47040 be returned to indicate the error.

47041 **ERRORS**

47042 The `posix_spawn_file_actions_init()` function shall fail if:

47043 [ENOMEM] Insufficient memory exists to initialize the spawn file actions object.

47044 The `posix_spawn_file_actions_destroy()` function may fail if:

47045 [EINVAL] The value specified by `file_actions` is invalid.

47046 **EXAMPLES**

47047 None.

47048 **APPLICATION USAGE**

47049 These functions are part of the Spawn option and need not be provided on all implementations.

47050 **RATIONALE**

47051 Refer to the RATIONALE in `posix_spawn_file_actions_addclose()`.

47052 **FUTURE DIRECTIONS**

47053 None.

47054 **SEE ALSO**

47055 `posix_spawn()`

47056 XBD `<spawn.h>`

CHANGE HISTORY

47057	CHANGE HISTORY
47058	First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
47059	In the SYNOPSIS, the inclusion of <code><sys/types.h></code> is no longer required.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_spawnattr_destroy()

System Interfaces

47060 **NAME**

47061 **posix_spawnattr_destroy, posix_spawnattr_init** — destroy and initialize spawn attributes object
 47062 **(ADVANCED REALTIME)**

47063 **SYNOPSIS**

```
47064 SPN      #include <spawn.h>
47065
47065      int posix_spawnattr_destroy(posix_spawnattr_t *attr);
47066      int posix_spawnattr_init(posix_spawnattr_t *attr);
```

47067 **DESCRIPTION**

47068 The *posix_spawnattr_destroy()* function shall destroy a spawn attributes object. A destroyed *attr*
 47069 attributes object can be reinitialized using *posix_spawnattr_init()*; the results of otherwise
 47070 referencing the object after it has been destroyed are undefined. An implementation may cause
 47071 *posix_spawnattr_destroy()* to set the object referenced by *attr* to an invalid value.

47072 The *posix_spawnattr_init()* function shall initialize a spawn attributes object *attr* with the default
 47073 value for all of the individual attributes used by the implementation. Results are undefined if
 47074 *posix_spawnattr_init()* is called specifying an already initialized *attr* attributes object.

47075 A spawn attributes object is of type **posix_spawnattr_t** (defined in **<spawn.h>**) and is used to
 47076 specify the inheritance of process attributes across a spawn operation. POSIX.1-2008 does not
 47077 define comparison or assignment operators for the type **posix_spawnattr_t**.

47078 Each implementation shall document the individual attributes it uses and their default values
 47079 unless these values are defined by POSIX.1-2008. Attributes not defined by POSIX.1-2008, their
 47080 default values, and the names of the associated functions to get and set those attribute values are
 47081 implementation-defined.

47082 The resulting spawn attributes object (possibly modified by setting individual attribute values),
 47083 is used to modify the behavior of *posix_spawn()* or *posix_spawnnp()*. After a spawn attributes
 47084 object has been used to spawn a process by a call to a *posix_spawn()* or *posix_spawnnp()*, any
 47085 function affecting the attributes object (including destruction) shall not affect any process that
 47086 has been spawned in this way.

47087 **RETURN VALUE**

47088 Upon successful completion, *posix_spawnattr_destroy()* and *posix_spawnattr_init()* shall return
 47089 zero; otherwise, an error number shall be returned to indicate the error.

47090 **ERRORS**

47091 The *posix_spawnattr_init()* function shall fail if:

47092 [ENOMEM] Insufficient memory exists to initialize the spawn attributes object.

47093 The *posix_spawnattr_destroy()* function may fail if:

47094 [EINVAL] The value specified by *attr* is invalid.

47095 **EXAMPLES**

47096 None.

47097 **APPLICATION USAGE**

47098 These functions are part of the Spawn option and need not be provided on all implementations.

47099 **RATIONALE**

47100 The original spawn interface proposed in POSIX.1-2008 defined the attributes that specify the
 47101 inheritance of process attributes across a spawn operation as a structure. In order to be able to
 47102 separate optional individual attributes under their appropriate options (that is, the *spawn-*
 47103 *schedparam* and *spawn-schedpolicy* attributes depending upon the Process Scheduling option), and

also for extensibility and consistency with the newer POSIX interfaces, the attributes interface has been changed to an opaque data type. This interface now consists of the type **posix_spawnattr_t**, representing a spawn attributes object, together with associated functions to initialize or destroy the attributes object, and to set or get each individual attribute. Although the new object-oriented interface is more verbose than the original structure, it is simple to use, more extensible, and easy to implement.

FUTURE DIRECTIONS

None.

SEE ALSO

[*posix_spawn\(\)*](#), [*posix_spawnattr_getsigdefault\(\)*](#), [*posix_spawnattr_getflags\(\)*](#),
[*posix_spawnattr_getpgroup\(\)*](#), [*posix_spawnattr_getschedparam\(\)*](#), [*posix_spawnattr_getschedpolicy\(\)*](#),
[*posix_spawnattr_getsigmask\(\)*](#)

XBD [*<spawn.h>*](#)

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #106 is applied, noting that the effect of initializing an already initialized spawn attributes option is undefined.

posix_spawnattr_getflags()

System Interfaces

47121 **NAME**

47122 `posix_spawnattr_getflags`, `posix_spawnattr_setflags` — get and set the spawn-flags attribute of a
 47123 spawn attributes object (**ADVANCED REALTIME**)

47124 **SYNOPSIS**

```
47125 SPN    #include <spawn.h>

47126    int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
47127                                short *restrict flags);
47128    int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

47129 **DESCRIPTION**

47130 The `posix_spawnattr_getflags()` function shall obtain the value of the *spawn-flags* attribute from the
 47131 attributes object referenced by *attr*.

47132 The `posix_spawnattr_setflags()` function shall set the *spawn-flags* attribute in an initialized
 47133 attributes object referenced by *attr*.

47134 The *spawn-flags* attribute is used to indicate which process attributes are to be changed in the
 47135 new process image when invoking `posix_spawn()` or `posix_spawnnp()`. It is the bitwise-inclusive
 47136 OR of zero or more of the following flags:

```
47137 POSIX_SPAWN_RESETIDS
47138 POSIX_SPAWN_SETPGROUP
47139 POSIX_SPAWN_SETSIGDEF
47140 POSIX_SPAWN_SETSIGMASK
47141 PS  POSIX_SPAWN_SETSCHEDPARAM
47142     POSIX_SPAWN_SETSCHEDULER
```

47143 These flags are defined in **<spawn.h>**. The default value of this attribute shall be as if no flags
 47144 were set.

47145 **RETURN VALUE**

47146 Upon successful completion, `posix_spawnattr_getflags()` shall return zero and store the value of
 47147 the *spawn-flags* attribute of *attr* into the object referenced by the *flags* parameter; otherwise, an
 47148 error number shall be returned to indicate the error.

47149 Upon successful completion, `posix_spawnattr_setflags()` shall return zero; otherwise, an error
 47150 number shall be returned to indicate the error.

47151 **ERRORS**

47152 These functions may fail if:

47153 [EINVAL] The value specified by *attr* is invalid.

47154 The `posix_spawnattr_setflags()` function may fail if:

47155 [EINVAL] The value of the attribute being set is not valid.

47156 **EXAMPLES**

47157 None.

47158 **APPLICATION USAGE**

47159 These functions are part of the Spawn option and need not be provided on all implementations.

47160 **RATIONALE**

47161 None.

47162 **FUTURE DIRECTIONS**

47163 None.

47164 **SEE ALSO**

47165 [*posix_spawn\(\)*](#), [*posix_spawnattr_destroy\(\)*](#), [*posix_spawnattr_getsigdefault\(\)*](#),
47166 [*posix_spawnattr_getpgroup\(\)*](#), [*posix_spawnattr_getschedparam\(\)*](#), [*posix_spawnattr_getschedpolicy\(\)*](#),
47167 [*posix_spawnattr_getsigmask\(\)*](#)

47168 XBD [*<spawn.h>*](#)47169 **CHANGE HISTORY**

47170 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

posix_spawnattr_getpgroup()

System Interfaces

47171 **NAME**

47172 `posix_spawnattr_getpgroup`, `posix_spawnattr_setpgroup` — get and set the spawn-pgroup
 47173 attribute of a spawn attributes object (**ADVANCED REALTIME**)

47174 **SYNOPSIS**

```
47175 SPN      #include <spawn.h>
47176
47176      int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,
47177                                   pid_t *restrict pgroup);
47177
47178      int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

47179 **DESCRIPTION**

47180 The `posix_spawnattr_getpgroup()` function shall obtain the value of the *spawn-pgroup* attribute
 47181 from the attributes object referenced by *attr*.

47182 The `posix_spawnattr_setpgroup()` function shall set the *spawn-pgroup* attribute in an initialized
 47183 attributes object referenced by *attr*.

47184 The *spawn-pgroup* attribute represents the process group to be joined by the new process image
 47185 in a spawn operation (if `POSIX_SPAWN_SETPGROUP` is set in the *spawn-flags* attribute). The
 47186 default value of this attribute shall be zero.

47187 **RETURN VALUE**

47188 Upon successful completion, `posix_spawnattr_getpgroup()` shall return zero and store the value of
 47189 the *spawn-pgroup* attribute of *attr* into the object referenced by the *pgroup* parameter; otherwise,
 47190 an error number shall be returned to indicate the error.

47191 Upon successful completion, `posix_spawnattr_setpgroup()` shall return zero; otherwise, an error
 47192 number shall be returned to indicate the error.

47193 **ERRORS**

47194 These functions may fail if:

47195 [EINVAL] The value specified by *attr* is invalid.

47196 The `posix_spawnattr_setpgroup()` function may fail if:

47197 [EINVAL] The value of the attribute being set is not valid.

47198 **EXAMPLES**

47199 None.

47200 **APPLICATION USAGE**

47201 These functions are part of the Spawn option and need not be provided on all implementations.

47202 **RATIONALE**

47203 None.

47204 **FUTURE DIRECTIONS**

47205 None.

47206 **SEE ALSO**

47207 `posix_spawn()`, `posix_spawnattr_destroy()`, `posix_spawnattr_getsigdefault()`,
 47208 `posix_spawnattr_getflags()`, `posix_spawnattr_getschedparam()`, `posix_spawnattr_getschedpolicy()`,
 47209 `posix_spawnattr_getsigmask()`

47210 XBD `<spawn.h>`

CHANGE HISTORY

47211 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
47212

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_spawnattr_getschedparam()

System Interfaces

47213 **NAME**

47214 **posix_spawnattr_getschedparam**, **posix_spawnattr_setschedparam** — get and set the spawn-
 47215 schedparam attribute of a spawn attributes object (**ADVANCED REALTIME**)

47216 **SYNOPSIS**

```
47217 SPN PS #include <spawn.h>
47218 #include <sched.h>

47219 int posix_spawnattr_getschedparam(const posix_spawnattr_t
47220     *restrict attr, struct sched_param *restrict schedparam);
47221 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
47222     const struct sched_param *restrict schedparam);
```

47223 **DESCRIPTION**

47224 The *posix_spawnattr_getschedparam()* function shall obtain the value of the *spawn-schedparam*
 47225 attribute from the attributes object referenced by *attr*.

47226 The *posix_spawnattr_setschedparam()* function shall set the *spawn-schedparam* attribute in an
 47227 initialized attributes object referenced by *attr*.

47228 The *spawn-schedparam* attribute represents the scheduling parameters to be assigned to the new
 47229 process image in a spawn operation (if **POSIX_SPAWN_SETSCHEDULER** or
 47230 **POSIX_SPAWN_SETSCHEDPARAM** is set in the *spawn-flags* attribute). The default value of this
 47231 attribute is unspecified.

47232 **RETURN VALUE**

47233 Upon successful completion, *posix_spawnattr_getschedparam()* shall return zero and store the
 47234 value of the *spawn-schedparam* attribute of *attr* into the object referenced by the *schedparam*
 47235 parameter; otherwise, an error number shall be returned to indicate the error.

47236 Upon successful completion, *posix_spawnattr_setschedparam()* shall return zero; otherwise, an
 47237 error number shall be returned to indicate the error.

47238 **ERRORS**

47239 These functions may fail if:

47240 [EINVAL] The value specified by *attr* is invalid.

47241 The *posix_spawnattr_setschedparam()* function may fail if:

47242 [EINVAL] The value of the attribute being set is not valid.

47243 **EXAMPLES**

47244 None.

47245 **APPLICATION USAGE**

47246 These functions are part of the Spawn and Process Scheduling options and need not be provided
 47247 on all implementations.

47248 **RATIONALE**

47249 None.

47250 **FUTURE DIRECTIONS**

47251 None.

47252 **SEE ALSO**

47253 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*,
 47254 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedpolicy()*,
 47255 *posix_spawnattr_getsigmask()*

47256 XBD [<sched.h>](#), [<spawn.h>](#)

47257 **CHANGE HISTORY**

47258 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_spawnattr_getschedpolicy()

System Interfaces

47259 **NAME**

47260 `posix_spawnattr_getschedpolicy`, `posix_spawnattr_setschedpolicy` — get and set the spawn-
 47261 `schedpolicy` attribute of a spawn attributes object (**ADVANCED REALTIME**)

47262 **SYNOPSIS**

```
47263 SPN PS #include <spawn.h>
47264 #include <sched.h>

47265 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t
47266     *restrict attr, int *restrict schedpolicy);
47267 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
47268     int schedpolicy);
```

47269 **DESCRIPTION**

47270 The `posix_spawnattr_getschedpolicy()` function shall obtain the value of the `spawn-schedpolicy`
 47271 attribute from the attributes object referenced by `attr`.

47272 The `posix_spawnattr_setschedpolicy()` function shall set the `spawn-schedpolicy` attribute in an
 47273 initialized attributes object referenced by `attr`.

47274 The `spawn-schedpolicy` attribute represents the scheduling policy to be assigned to the new
 47275 process image in a spawn operation (if `POSIX_SPAWN_SETSCHEDULER` is set in the `spawn-`
 47276 `flags` attribute). The default value of this attribute is unspecified.

47277 **RETURN VALUE**

47278 Upon successful completion, `posix_spawnattr_getschedpolicy()` shall return zero and store the
 47279 value of the `spawn-schedpolicy` attribute of `attr` into the object referenced by the `schedpolicy`
 47280 parameter; otherwise, an error number shall be returned to indicate the error.

47281 Upon successful completion, `posix_spawnattr_setschedpolicy()` shall return zero; otherwise, an
 47282 error number shall be returned to indicate the error.

47283 **ERRORS**

47284 These functions may fail if:

47285 [EINVAL] The value specified by `attr` is invalid.

47286 The `posix_spawnattr_setschedpolicy()` function may fail if:

47287 [EINVAL] The value of the attribute being set is not valid.

47288 **EXAMPLES**

47289 None.

47290 **APPLICATION USAGE**

47291 These functions are part of the Spawn and Process Scheduling options and need not be provided
 47292 on all implementations.

47293 **RATIONALE**

47294 None.

47295 **FUTURE DIRECTIONS**

47296 None.

47297 **SEE ALSO**

47298 `posix_spawn()`, `posix_spawnattr_destroy()`, `posix_spawnattr_getsigdefault()`,
 47299 `posix_spawnattr_getflags()`, `posix_spawnattr_getpgroup()`, `posix_spawnattr_getschedparam()`,
 47300 `posix_spawnattr_getsigmask()`

47301 XBD `<sched.h>`, `<spawn.h>`

CHANGE HISTORY

47302
47303 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_spawnattr_getsigdefault()

System Interfaces

47304 **NAME**

47305 `posix_spawnattr_getsigdefault`, `posix_spawnattr_setsigdefault` — get and set the `spawn-sigdefault`
 47306 attribute of a spawn attributes object (**ADVANCED REALTIME**)

47307 **SYNOPSIS**

```
47308 #include <signal.h>
47309 #include <spawn.h>

47310 int posix_spawnattr_getsigdefault(const posix_spawnattr_t
47311     *restrict attr, sigset_t *restrict sigdefault);
47312 int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
47313     const sigset_t *restrict sigdefault);
```

47314 **DESCRIPTION**

47315 The `posix_spawnattr_getsigdefault()` function shall obtain the value of the `spawn-sigdefault`
 47316 attribute from the attributes object referenced by `attr`.

47317 The `posix_spawnattr_setsigdefault()` function shall set the `spawn-sigdefault` attribute in an
 47318 initialized attributes object referenced by `attr`.

47319 The `spawn-sigdefault` attribute represents the set of signals to be forced to default signal handling
 47320 in the new process image (if `POSIX_SPAWN_SETSIGDEF` is set in the `spawn-flags` attribute) by a
 47321 spawn operation. The default value of this attribute shall be an empty signal set.

47322 **RETURN VALUE**

47323 Upon successful completion, `posix_spawnattr_getsigdefault()` shall return zero and store the value
 47324 of the `spawn-sigdefault` attribute of `attr` into the object referenced by the `sigdefault` parameter;
 47325 otherwise, an error number shall be returned to indicate the error.

47326 Upon successful completion, `posix_spawnattr_setsigdefault()` shall return zero; otherwise, an error
 47327 number shall be returned to indicate the error.

47328 **ERRORS**

47329 These functions may fail if:

47330 [EINVAL] The value specified by `attr` is invalid.

47331 The `posix_spawnattr_setsigdefault()` function may fail if:

47332 [EINVAL] The value of the attribute being set is not valid.

47333 **EXAMPLES**

47334 None.

47335 **APPLICATION USAGE**

47336 These functions are part of the Spawn option and need not be provided on all implementations.

47337 **RATIONALE**

47338 None.

47339 **FUTURE DIRECTIONS**

47340 None.

47341 **SEE ALSO**

47342 `posix_spawn()`, `posix_spawnattr_destroy()`, `posix_spawnattr_getflags()`, `posix_spawnattr_getpgroup()`,
 47343 `posix_spawnattr_getschedparam()`, `posix_spawnattr_getschedpolicy()`, `posix_spawnattr_getsigmask()`

47344 XBD `<signal.h>`, `<spawn.h>`

CHANGE HISTORY

47345 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
47346

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_spawnattr_getsigmask()

System Interfaces

47347 **NAME**

47348 **posix_spawnattr_getsigmask**, **posix_spawnattr_setsigmask** — get and set the spawn-sigmask
 47349 attribute of a spawn attributes object (**ADVANCED REALTIME**)

47350 **SYNOPSIS**

```
47351 SPN      #include <signal.h>
47352           #include <spawn.h>

47353           int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
47354                                           sigset_t *restrict sigmask);
47355           int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
47356                                           const sigset_t *restrict sigmask);
```

47357 **DESCRIPTION**

47358 The *posix_spawnattr_getsigmask()* function shall obtain the value of the *spawn-sigmask* attribute
 47359 from the attributes object referenced by *attr*.

47360 The *posix_spawnattr_setsigmask()* function shall set the *spawn-sigmask* attribute in an initialized
 47361 attributes object referenced by *attr*.

47362 The *spawn-sigmask* attribute represents the signal mask in effect in the new process image of a
 47363 spawn operation (if **POSIX_SPAWN_SETSIGMASK** is set in the *spawn-flags* attribute). The
 47364 default value of this attribute is unspecified.

47365 **RETURN VALUE**

47366 Upon successful completion, *posix_spawnattr_getsigmask()* shall return zero and store the value
 47367 of the *spawn-sigmask* attribute of *attr* into the object referenced by the *sigmask* parameter;
 47368 otherwise, an error number shall be returned to indicate the error.

47369 Upon successful completion, *posix_spawnattr_setsigmask()* shall return zero; otherwise, an error
 47370 number shall be returned to indicate the error.

47371 **ERRORS**

47372 These functions may fail if:

47373 [EINVAL] The value specified by *attr* is invalid.

47374 The *posix_spawnattr_setsigmask()* function may fail if:

47375 [EINVAL] The value of the attribute being set is not valid.

47376 **EXAMPLES**

47377 None.

47378 **APPLICATION USAGE**

47379 These functions are part of the Spawn option and need not be provided on all implementations.

47380 **RATIONALE**

47381 None.

47382 **FUTURE DIRECTIONS**

47383 None.

47384 **SEE ALSO**

47385 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*,
 47386 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*,
 47387 *posix_spawnattr_getschedpolicy()*

47388 XBD **<signal.h>**, **<spawn.h>**

CHANGE HISTORY

47389
47390 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_spawnattr_init()*System Interfaces*47391 **NAME**47392 posix_spawnattr_init — initialize the spawn attributes object (**ADVANCED REALTIME**)47393 **SYNOPSIS**47394 SPN `#include <spawn.h>`47395 `int posix_spawnattr_init(posix_spawnattr_t *attr);`47396 **DESCRIPTION**47397 Refer to *posix_spawnattr_destroy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***posix_spawnattr_setflags()**47398 **NAME**

47399 posix_spawnattr_setflags — set the spawn-flags attribute of a spawn attributes object
47400 (**ADVANCED REALTIME**)

47401 **SYNOPSIS**

```
47402 SPN        #include <spawn.h>  
47403        int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

47404 **DESCRIPTION**

47405 Refer to *posix_spawnattr_getflags()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_spawnattr_setpgroup()*System Interfaces*47406 **NAME**

47407 `posix_spawnattr_setpgroup` — set the spawn-pgroup attribute of a spawn attributes object
47408 (**ADVANCED REALTIME**)

47409 **SYNOPSIS**

47410 SPN `#include <spawn.h>`

47411 `int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);`

47412 **DESCRIPTION**

47413 Refer to [*posix_spawnattr_getpgroup\(\)*](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***posix_spawnattr_setschedparam()**47414 **NAME**

47415 **posix_spawnattr_setschedparam** — set the spawn-schedparam attribute of a spawn attributes
47416 object (**ADVANCED REALTIME**)

47417 **SYNOPSIS**

```
47418 SPN PS #include <sched.h>  
47419         #include <spawn.h>  
  
47420         int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,  
47421         const struct sched_param *restrict schedparam);
```

47422 **DESCRIPTION**

47423 Refer to [posix_spawnattr_getschedparam\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_spawnattr_setschedpolicy()*System Interfaces*47424 **NAME**

47425 **posix_spawnattr_setschedpolicy** — set the spawn-schedpolicy attribute of a spawn attributes
47426 object (**ADVANCED REALTIME**)

47427 **SYNOPSIS**

```
47428 SPN PS  #include <sched.h>  
47429          #include <spawn.h>  
  
47430          int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,  
47431          int schedpolicy);
```

47432 **DESCRIPTION**

47433 Refer to [posix_spawnattr_getschedpolicy\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***posix_spawnattr_setsigdefault()**47434 **NAME**

47435 **posix_spawnattr_setsigdefault** — set the spawn-sigdefault attribute of a spawn attributes object
47436 (**ADVANCED REALTIME**)

47437 **SYNOPSIS**

```
47438 SPN        #include <signal.h>  
47439        #include <spawn.h>  
  
47440        int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,  
47441        const sigset_t *restrict sigdefault);
```

47442 **DESCRIPTION**

47443 Refer to [posix_spawnattr_getsigdefault\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_spawnattr_setsigmask()*System Interfaces*47444 **NAME**

47445 **posix_spawnattr_setsigmask** — set the spawn-sigmask attribute of a spawn attributes object
47446 (**ADVANCED REALTIME**)

47447 **SYNOPSIS**

```
47448 SPN        #include <signal.h>  
47449        #include <spawn.h>  
  
47450        int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,  
47451        const sigset_t *restrict sigmask);
```

47452 **DESCRIPTION**

47453 Refer to [posix_spawnattr_getsigmask\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***posix_spawn()**47454 **NAME**47455 **posix_spawn** — spawn a process (**ADVANCED REALTIME**)47456 **SYNOPSIS**

```
47457 SPN      #include <spawn.h>
47458
47458      int posix_spawn(pid_t *restrict pid, const char *restrict file,
47459                    const posix_spawn_file_actions_t *file_actions,
47460                    const posix_spawnattr_t *restrict attrp,
47461                    char *const argv[restrict], char *const envp[restrict]);
```

47462 **DESCRIPTION**47463 Refer to *posix_spawn()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_attr_destroy()

System Interfaces

47464 **NAME**

47465 `posix_trace_attr_destroy`, `posix_trace_attr_init` — destroy and initialize the trace stream
 47466 attributes object (**TRACING**)

47467 **SYNOPSIS**

```
47468 OB TRC #include <trace.h>
47469
47469 int posix_trace_attr_destroy(trace_attr_t *attr);
47470 int posix_trace_attr_init(trace_attr_t *attr);
```

47471 **DESCRIPTION**

47472 The `posix_trace_attr_destroy()` function shall destroy an initialized trace attributes object. A
 47473 destroyed `attr` attributes object can be reinitialized using `posix_trace_attr_init()`; the results of
 47474 otherwise referencing the object after it has been destroyed are undefined.

47475 The `posix_trace_attr_init()` function shall initialize a trace attributes object `attr` with the default
 47476 value for all of the individual attributes used by a given implementation. The read-only
 47477 *generation-version* and *clock-resolution* attributes of the newly initialized trace attributes object
 47478 shall be set to their appropriate values (see [Section 2.11.1.2](#), on page 535).

47479 Results are undefined if `posix_trace_attr_init()` is called specifying an already initialized `attr`
 47480 attributes object.

47481 Implementations may add extensions to the trace attributes object structure as permitted in XBD
 47482 [Chapter 2](#) (on page 15).

47483 The resulting attributes object (possibly modified by setting individual attributes values), when
 47484 used by `posix_trace_create()`, defines the attributes of the trace stream created. A single attributes
 47485 object can be used in multiple calls to `posix_trace_create()`. After one or more trace streams have
 47486 been created using an attributes object, any function affecting that attributes object, including
 47487 destruction, shall not affect any trace stream previously created. An initialized attributes object
 47488 also serves to receive the attributes of an existing trace stream or trace log when calling the
 47489 `posix_trace_get_attr()` function.

47490 **RETURN VALUE**

47491 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 47492 return the corresponding error number.

47493 **ERRORS**

47494 The `posix_trace_attr_destroy()` function may fail if:

47495 [EINVAL] The value of `attr` is invalid.

47496 The `posix_trace_attr_init()` function shall fail if:

47497 [ENOMEM] Insufficient memory exists to initialize the trace attributes object.

47498 **EXAMPLES**

47499 None.

47500 **APPLICATION USAGE**

47501 None.

47502 **RATIONALE**

47503 None.

47504 **FUTURE DIRECTIONS**

47505 The *posix_trace_attr_destroy()* and *posix_trace_attr_init()* functions may be removed in a future
47506 version.

47507 **SEE ALSO**

47508 *posix_trace_create()*, *posix_trace_get_attr()*, *uname()*

47509 XBD <trace.h>

47510 **CHANGE HISTORY**

47511 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

47512 IEEE PASC Interpretation 1003.1 #123 is applied.

47513 **Issue 7**

47514 The *posix_trace_attr_destroy()* and *posix_trace_attr_init()* functions are marked obsolescent.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_attr_getclockres()

System Interfaces

47515 **NAME**

47516 `posix_trace_attr_getclockres`, `posix_trace_attr_getcreatetime`, `posix_trace_attr_getgenversion`,
 47517 `posix_trace_attr_getname`, `posix_trace_attr_setname` — retrieve and set information about a
 47518 trace stream (**TRACING**)

47519 **SYNOPSIS**

```
47520 OB TRC #include <time.h>
47521 #include <trace.h>

47522 int posix_trace_attr_getclockres(const trace_attr_t *attr,
47523     struct timespec *resolution);
47524 int posix_trace_attr_getcreatetime(const trace_attr_t *attr,
47525     struct timespec *createtime);

47526 #include <trace.h>

47527 int posix_trace_attr_getgenversion(const trace_attr_t *attr,
47528     char *genversion);
47529 int posix_trace_attr_getname(const trace_attr_t *attr,
47530     char *tracename);
47531 int posix_trace_attr_setname(trace_attr_t *attr,
47532     const char *tracename);
```

47533 **DESCRIPTION**

47534 The `posix_trace_attr_getclockres()` function shall copy the clock resolution of the clock used to
 47535 generate timestamps from the *clock-resolution* attribute of the attributes object pointed to by the
 47536 *attr* argument into the structure pointed to by the *resolution* argument.

47537 The `posix_trace_attr_getcreatetime()` function shall copy the trace stream creation time from the
 47538 *creation-time* attribute of the attributes object pointed to by the *attr* argument into the structure
 47539 pointed to by the *createtime* argument. The *creation-time* attribute shall represent the time of
 47540 creation of the trace stream.

47541 The `posix_trace_attr_getgenversion()` function shall copy the string containing version information
 47542 from the *generation-version* attribute of the attributes object pointed to by the *attr* argument into
 47543 the string pointed to by the *genversion* argument. The *genversion* argument shall be the address of
 47544 a character array which can store at least {TRACE_NAME_MAX} characters.

47545 The `posix_trace_attr_getname()` function shall copy the string containing the trace name from the
 47546 *trace-name* attribute of the attributes object pointed to by the *attr* argument into the string
 47547 pointed to by the *tracename* argument. The *tracename* argument shall be the address of a character
 47548 array which can store at least {TRACE_NAME_MAX} characters.

47549 The `posix_trace_attr_setname()` function shall set the name in the *trace-name* attribute of the
 47550 attributes object pointed to by the *attr* argument, using the trace name string supplied by the
 47551 *tracename* argument. If the supplied string contains more than {TRACE_NAME_MAX}
 47552 characters, the name copied into the *trace-name* attribute may be truncated to one less than the
 47553 length of {TRACE_NAME_MAX} characters. The default value is a null string.

47554 **RETURN VALUE**

47555 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 47556 return the corresponding error number.

47557 If successful, the `posix_trace_attr_getclockres()` function stores the *clock-resolution* attribute value in
 47558 the object pointed to by *resolution*. Otherwise, the content of this object is unspecified.

47559 If successful, the `posix_trace_attr_getcreatetime()` function stores the trace stream creation time in

47560 the object pointed to by *createtime*. Otherwise, the content of this object is unspecified.

47561 If successful, the *posix_trace_attr_getgenversion()* function stores the trace version information in
47562 the string pointed to by *genversion*. Otherwise, the content of this string is unspecified.

47563 If successful, the *posix_trace_attr_getname()* function stores the trace name in the string pointed
47564 to by *tracename*. Otherwise, the content of this string is unspecified.

47565 **ERRORS**

47566 The *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
47567 and *posix_trace_attr_getname()* functions may fail if:

47568 [EINVAL] The value specified by one of the arguments is invalid.

47569 **EXAMPLES**

47570 None.

47571 **APPLICATION USAGE**

47572 None.

47573 **RATIONALE**

47574 None.

47575 **FUTURE DIRECTIONS**

47576 The *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
47577 *posix_trace_attr_getname()*, and *posix_trace_attr_setname()* functions may be removed in a future
47578 version.

47579 **SEE ALSO**

47580 *posix_trace_attr_destroy()*, *posix_trace_create()*, *posix_trace_get_attr()*, *uname()*

47581 XBD [<time.h>](#), [<trace.h>](#)

47582 **CHANGE HISTORY**

47583 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

47584 **Issue 7**

47585 The *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
47586 *posix_trace_attr_getname()*, and *posix_trace_attr_setname()* functions are marked obsolescent.

posix_trace_attr_getinherited()

System Interfaces

47587 **NAME**

47588 `posix_trace_attr_getinherited`, `posix_trace_attr_getlogfullpolicy`,
 47589 `posix_trace_attr_getstreamfullpolicy`, `posix_trace_attr_setinherited`,
 47590 `posix_trace_attr_setlogfullpolicy`, `posix_trace_attr_setstreamfullpolicy` — retrieve and set the
 47591 behavior of a trace stream (**TRACING**)

47592 **SYNOPSIS**

```
47593 OB TRC  #include <trace.h>

47594 TRI      int posix_trace_attr_getinherited(const trace_attr_t *restrict attr,
47595                                             int *restrict inheritancepolicy);
47596 TRL      int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict attr,
47597                                                 int *restrict logpolicy);
47598          int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict
47599                                                  attr, int *restrict streampolicy);
47600 TRI      int posix_trace_attr_setinherited(trace_attr_t *attr,
47601                                             int inheritancepolicy);
47602 TRL      int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
47603                                                 int logpolicy);
47604          int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
47605                                                  int streampolicy);
```

47606 **DESCRIPTION**

47607 TRI The `posix_trace_attr_getinherited()` and `posix_trace_attr_setinherited()` functions, respectively, shall
 47608 get and set the inheritance policy stored in the *inheritance* attribute for traced processes across the
 47609 `fork()` and `spawn()` operations. The *inheritance* attribute of the attributes object pointed to by the
 47610 *attr* argument shall be set to one of the following values defined by manifest constants in the
 47611 **<trace.h>** header:

47612 **POSIX_TRACE_CLOSE_FOR_CHILD**

47613 After a `fork()` or `spawn()` operation, the child shall not be traced, and tracing of the parent
 47614 shall continue.

47615 **POSIX_TRACE_INHERITED**

47616 After a `fork()` or `spawn()` operation, if the parent is being traced, its child shall be
 47617 concurrently traced using the same trace stream.

47618 The default value for the *inheritance* attribute is **POSIX_TRACE_CLOSE_FOR_CHILD**.

47619 TRL The `posix_trace_attr_getlogfullpolicy()` and `posix_trace_attr_setlogfullpolicy()` functions,
 47620 respectively, shall get and set the trace log full policy stored in the *log-full-policy* attribute of the
 47621 attributes object pointed to by the *attr* argument.

47622 The *log-full-policy* attribute shall be set to one of the following values defined by manifest
 47623 constants in the **<trace.h>** header:

47624 **POSIX_TRACE_LOOP**

47625 The trace log shall loop until the associated trace stream is stopped. This policy means that
 47626 when the trace log gets full, the file system shall reuse the resources allocated to the oldest
 47627 trace events that were recorded. In this way, the trace log will always contain the most
 47628 recent trace events flushed.

47629 **POSIX_TRACE_UNTIL_FULL**

47630 The trace stream shall be flushed to the trace log until the trace log is full. This condition can
 47631 be deduced from the *posix_log_full_status* member status (see the **posix_trace_status_info**
 47632 structure defined in **<trace.h>**). The last recorded trace event shall be the
 47633 **POSIX_TRACE_STOP** trace event.

POSIX_TRACE_APPEND

The associated trace stream shall be flushed to the trace log without log size limitation. If the application specifies `POSIX_TRACE_APPEND`, the implementation shall ignore the *log-max-size* attribute.

The default value for the *log-full-policy* attribute is `POSIX_TRACE_LOOP`.

The `posix_trace_attr_getstreamfullpolicy()` and `posix_trace_attr_setstreamfullpolicy()` functions, respectively, shall get and set the trace stream full policy stored in the *stream-full-policy* attribute of the attributes object pointed to by the *attr* argument.

The *stream-full-policy* attribute shall be set to one of the following values defined by manifest constants in the `<trace.h>` header:

POSIX_TRACE_LOOP

The trace stream shall loop until explicitly stopped by the `posix_trace_stop()` function. This policy means that when the trace stream is full, the trace system shall reuse the resources allocated to the oldest trace events recorded. In this way, the trace stream will always contain the most recent trace events recorded.

POSIX_TRACE_UNTIL_FULL

The trace stream will run until the trace stream resources are exhausted. Then the trace stream will stop. This condition can be deduced from `posix_stream_status` and `posix_stream_full_status` (see the `posix_trace_status_info` structure defined in `<trace.h>`). When this trace stream is read, a `POSIX_TRACE_STOP` trace event shall be reported after reporting the last recorded trace event. The trace system shall reuse the resources allocated to any trace events already reported—see the `posix_trace_getnext_event()`, `posix_trace_trygetnext_event()`, and `posix_trace_timedgetnext_event()` functions—or already flushed for an active trace stream with log if the Trace Log option is supported; see the `posix_trace_flush()` function. The trace system shall restart the trace stream when it is empty and may restart it sooner. A `POSIX_TRACE_START` trace event shall be reported before reporting the next recorded trace event.

POSIX_TRACE_FLUSH

If the Trace Log option is supported, this policy is identical to the `POSIX_TRACE_UNTIL_FULL` trace stream full policy except that the trace stream shall be flushed regularly as if `posix_trace_flush()` had been explicitly called. Defining this policy for an active trace stream without log shall be invalid.

The default value for the *stream-full-policy* attribute shall be `POSIX_TRACE_LOOP` for an active trace stream without log.

If the Trace Log option is supported, the default value for the *stream-full-policy* attribute shall be `POSIX_TRACE_FLUSH` for an active trace stream with log.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

If successful, the `posix_trace_attr_getinherited()` function shall store the *inheritance* attribute value in the object pointed to by *inheritancepolicy*. Otherwise, the content of this object is undefined.

If successful, the `posix_trace_attr_getlogfullpolicy()` function shall store the *log-full-policy* attribute value in the object pointed to by *logpolicy*. Otherwise, the content of this object is undefined.

If successful, the `posix_trace_attr_getstreamfullpolicy()` function shall store the *stream-full-policy* attribute value in the object pointed to by *streampolicy*. Otherwise, the content of this object is undefined.

posix_trace_attr_getinherited()

System Interfaces

47680 **ERRORS**

47681 These functions may fail if:

47682 [EINVAL] The value specified by at least one of the arguments is invalid.

47683 **EXAMPLES**

47684 None.

47685 **APPLICATION USAGE**

47686 None.

47687 **RATIONALE**

47688 None.

47689 **FUTURE DIRECTIONS**

47690 The following functions:

47691 *posix_trace_attr_getinherited()*
 47692 *posix_trace_attr_getlogfullpolicy()*
 47693 *posix_trace_attr_getstreamfullpolicy()*
 47694 *posix_trace_attr_setinherited()*
 47695 *posix_trace_attr_setlogfullpolicy()*
 47696 *posix_trace_attr_setstreamfullpolicy()*

47697 may be removed in a future version.

47698 **SEE ALSO**

47699 *fork()*, *posix_trace_attr_destroy()*, *posix_trace_create()*, *posix_trace_get_attr()*,
 47700 *posix_trace_getnext_event()*, *posix_trace_start()*

47701 XBD <trace.h>

47702 **CHANGE HISTORY**

47703 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

47704 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/39 is applied, adding the TRL and TRC
 47705 margin codes to the *posix_trace_attr_setlogfullpolicy()* function.

47706 **Issue 7**

47707 SD5-XSH-ERN-116 is applied, adding the missing **restrict** keyword to the
 47708 *posix_trace_attr_getstreamfullpolicy()* function declaration.

47709 These functions are marked obsolescent.

System Interfaces

posix_trace_attr_getlogsize()**NAME**

posix_trace_attr_getlogsize, posix_trace_attr_getmaxdatasize,
 posix_trace_attr_getmaxsystemeventsizesize, posix_trace_attr_getmaxusereventsizesize,
 posix_trace_attr_getstreamsize, posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize,
 posix_trace_attr_setstreamsize — retrieve and set trace stream size attributes (TRACING)

SYNOPSIS

```
OB TRC #include <sys/types.h>
        #include <trace.h>

TRL int posix_trace_attr_getlogsize(const trace_attr_t *restrict attr,
    size_t *restrict logsize);

int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict attr,
    size_t *restrict maxdatasize);

int posix_trace_attr_getmaxsystemeventsizesize(
    const trace_attr_t *restrict attr,
    size_t *restrict eventsizesize);

int posix_trace_attr_getmaxusereventsizesize(
    const trace_attr_t *restrict attr,
    size_t data_len, size_t *restrict eventsizesize);

int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
    size_t *restrict streamsize);

TRL int posix_trace_attr_setlogsize(trace_attr_t *attr,
    size_t logsize);

int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
    size_t maxdatasize);

int posix_trace_attr_setstreamsize(trace_attr_t *attr,
    size_t streamsize);
```

DESCRIPTION

The *posix_trace_attr_getlogsize()* function shall copy the log size, in bytes, from the *log-max-size* attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by the *logsize* argument. This log size is the maximum total of bytes that shall be allocated for system and user trace events in the trace log. The default value for the *log-max-size* attribute is implementation-defined.

The *posix_trace_attr_setlogsize()* function shall set the maximum allowed size, in bytes, in the *log-max-size* attribute of the attributes object pointed to by the *attr* argument, using the size value supplied by the *logsize* argument.

The trace log size shall be used if the *log-full-policy* attribute is set to *POSIX_TRACE_LOOP* or *POSIX_TRACE_UNTIL_FULL*. If the *log-full-policy* attribute is set to *POSIX_TRACE_APPEND*, the implementation shall ignore the *log-max-size* attribute.

The *posix_trace_attr_getmaxdatasize()* function shall copy the maximum user trace event data size, in bytes, from the *max-data-size* attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by the *maxdatasize* argument. The default value for the *max-data-size* attribute is implementation-defined.

The *posix_trace_attr_getmaxsystemeventsizesize()* function shall calculate the maximum memory size, in bytes, required to store a single system trace event. This value is calculated for the trace stream attributes object pointed to by the *attr* argument and is returned in the variable pointed to by the *eventsizesize* argument.

The values returned as the maximum memory sizes of the user and system trace events shall be such that if the sum of the maximum memory sizes of a set of the trace events that may be

posix_trace_attr_getlogsize()

System Interfaces

recorded in a trace stream is less than or equal to the *stream-min-size* attribute of that trace stream, the system provides the necessary resources for recording all those trace events, without loss.

The *posix_trace_attr_getmaxusereventsize()* function shall calculate the maximum memory size, in bytes, required to store a single user trace event generated by a call to *posix_trace_event()* with a *data_len* parameter equal to the *data_len* value specified in this call. This value is calculated for the trace stream attributes object pointed to by the *attr* argument and is returned in the variable pointed to by the *eventsize* argument.

The *posix_trace_attr_getstreamsize()* function shall copy the stream size, in bytes, from the *stream-min-size* attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by the *streamsize* argument.

This stream size is the current total memory size reserved for system and user trace events in the trace stream. The default value for the *stream-min-size* attribute is implementation-defined. The stream size refers to memory used to store trace event records. Other stream data (for example, trace attribute values) shall not be included in this size.

The *posix_trace_attr_setmaxdatasize()* function shall set the maximum allowed size, in bytes, in the *max-data-size* attribute of the attributes object pointed to by the *attr* argument, using the size value supplied by the *maxdatasize* argument. This maximum size is the maximum allowed size for the user data argument which may be passed to *posix_trace_event()*. The implementation shall be allowed to truncate data passed to *trace_user_event* which is longer than *maxdatasize*.

The *posix_trace_attr_setstreamsize()* function shall set the minimum allowed size, in bytes, in the *stream-min-size* attribute of the attributes object pointed to by the *attr* argument, using the size value supplied by the *streamsize* argument.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

TRL The *posix_trace_attr_getlogsize()* function stores the maximum trace log allowed size in the object pointed to by *logsize*, if successful.

The *posix_trace_attr_getmaxdatasize()* function stores the maximum trace event record memory size in the object pointed to by *maxdatasize*, if successful.

The *posix_trace_attr_getmaxsystemeventsize()* function stores the maximum memory size to store a single system trace event in the object pointed to by *eventsize*, if successful.

The *posix_trace_attr_getmaxusereventsize()* function stores the maximum memory size to store a single user trace event in the object pointed to by *eventsize*, if successful.

The *posix_trace_attr_getstreamsize()* function stores the maximum trace stream allowed size in the object pointed to by *streamsize*, if successful.

ERRORS

These functions may fail if:

[EINVAL] The value specified by one of the arguments is invalid.

System Interfaces

posix_trace_attr_getlogsize()47797 **EXAMPLES**

47798 None.

47799 **APPLICATION USAGE**

47800 None.

47801 **RATIONALE**

47802 None.

47803 **FUTURE DIRECTIONS**

47804 The following functions:

47805 `posix_trace_attr_getlogsize()`
 47806 `posix_trace_attr_getmaxdatasize()`
 47807 `posix_trace_attr_getmaxsystemeventszsize()`
 47808 `posix_trace_attr_getmaxusereventszsize()`
 47809 `posix_trace_attr_getstreamsize()`
 47810 `posix_trace_attr_setlogsize()`
 47811 `posix_trace_attr_setmaxdatasize()`
 47812 `posix_trace_attr_setstreamsize()`

47813 may be removed in a future version.

47814 **SEE ALSO**47815 `posix_trace_attr_destroy()`, `posix_trace_create()`, `posix_trace_event()`, `posix_trace_get_attr()`47816 XBD `<sys/types.h>`, `<trace.h>`47817 **CHANGE HISTORY**

47818 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

47819 **Issue 7**

47820 These functions are marked obsolescent.

posix_trace_attr_getname()*System Interfaces*47821 **NAME**47822 posix_trace_attr_getname — retrieve and set information about a trace stream (**TRACING**)47823 **SYNOPSIS**

```
47824 OB TRC #include <trace.h>
47825         int posix_trace_attr_getname(const trace_attr_t *attr,
47826         char *tracename);
```

47827 **DESCRIPTION**47828 Refer to [posix_trace_attr_getclockres\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***posix_trace_attr_getstreamfullpolicy()**47829 **NAME**

47830 **posix_trace_attr_getstreamfullpolicy** — retrieve and set the behavior of a trace stream
47831 (**TRACING**)

47832 **SYNOPSIS**

```
47833 OB TRC #include <trace.h>  
47834  
47835 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict  
47836 attr, int *restrict streampolicy);
```

47836 **DESCRIPTION**

47837 Refer to [posix_trace_attr_getinherited\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_attr_getstreamsize()*System Interfaces*47838 **NAME**

47839 posix_trace_attr_getstreamsize — retrieve and set trace stream size attributes (TRACING)

47840 **SYNOPSIS**

```
47841 OB TRC #include <sys/types.h>
47842         #include <trace.h>
47843         int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
47844         size_t *restrict streamsize);
```

47845 **DESCRIPTION**47846 Refer to [posix_trace_attr_getlogsize\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***posix_trace_attr_init()**47847 **NAME**

47848 posix_trace_attr_init — initialize the trace stream attributes object (TRACING)

47849 **SYNOPSIS**47850 OB TRC `#include <trace.h>`47851 `int posix_trace_attr_init(trace_attr_t *attr);`47852 **DESCRIPTION**47853 Refer to *posix_trace_attr_destroy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_attr_setinherited()*System Interfaces*47854 **NAME**

47855 posix_trace_attr_setinherited, posix_trace_attr_setlogfullpolicy — retrieve and set the behavior
47856 of a trace stream (**TRACING**)

47857 **SYNOPSIS**

```
47858 OB TRC  #include <trace.h>
47859 TRI      int posix_trace_attr_setinherited(trace_attr_t *attr,
47860      int inheritance_policy);
47861 TRL      int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
47862      int log_policy);
```

47863 **DESCRIPTION**

47864 Refer to [posix_trace_attr_getinherited\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***posix_trace_attr_setlogsize()**47865 **NAME**

47866 `posix_trace_attr_setlogsize`, `posix_trace_attr_setmaxdatasize` — retrieve and set trace stream size
47867 attributes (**TRACING**)

47868 **SYNOPSIS**

```
47869 OB TRC  #include <sys/types.h>  
47870          #include <trace.h>  
  
47871 TRL      int posix_trace_attr_setlogsize(trace_attr_t *attr,  
47872          size_t logsize);  
47873 OB TRC  int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,  
47874          size_t maxdatasize);
```

47875 **DESCRIPTION**

47876 Refer to [*posix_trace_attr_getlogsize\(\)*](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_attr_setname()*System Interfaces*47877 **NAME**47878 posix_trace_attr_setname — retrieve and set information about a trace stream (**TRACING**)47879 **SYNOPSIS**

```
47880 OB TRC #include <trace.h>
47881         int posix_trace_attr_setname(trace_attr_t *attr,
47882         const char *tracename);
```

47883 **DESCRIPTION**47884 Refer to *posix_trace_attr_getclockres()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***posix_trace_attr_setstreamfullpolicy()**47885 **NAME**

47886 **posix_trace_attr_setstreamfullpolicy** — retrieve and set the behavior of a trace stream
47887 (**TRACING**)

47888 **SYNOPSIS**

```
47889 OB TRC #include <trace.h>  
47890  
47890        int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,  
47891        int streampolicy);
```

47892 **DESCRIPTION**

47893 Refer to [posix_trace_attr_getinherited\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_attr_setstreamsize()*System Interfaces*47894 **NAME**47895 posix_trace_attr_setstreamsize — retrieve and set trace stream size attributes (**TRACING**)47896 **SYNOPSIS**

```
47897 OB TRC #include <sys/types.h>  
47898         #include <trace.h>  
  
47899         int posix_trace_attr_setstreamsize(trace_attr_t *attr,  
47900         size_t streamsize);
```

47901 **DESCRIPTION**47902 Refer to [posix_trace_attr_getlogsize\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

posix_trace_clear()**NAME**

posix_trace_clear — clear trace stream and trace log (TRACING)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_clear(trace_id_t trid);
```

DESCRIPTION

The *posix_trace_clear()* function shall reinitialize the trace stream identified by the argument *trid* as if it were returning from the *posix_trace_create()* function, except that the same allocated resources shall be reused, the mapping of trace event type identifiers to trace event names shall be unchanged, and the trace stream status shall remain unchanged (that is, if it was running, it remains running and if it was suspended, it remains suspended).

All trace events in the trace stream recorded before the call to *posix_trace_clear()* shall be lost. The *posix_stream_full_status* status shall be set to *POSIX_TRACE_NOT_FULL*. There is no guarantee that all trace events that occurred during the *posix_trace_clear()* call are recorded; the behavior with respect to trace points that may occur during this call is unspecified.

If the Trace Log option is supported and the trace stream has been created with a log, the *posix_trace_clear()* function shall reinitialize the trace stream with the same behavior as if the trace stream was created without the log, plus it shall reinitialize the trace log associated with the trace stream identified by the argument *trid* as if it were returning from the *posix_trace_create_withlog()* function, except that the same allocated resources, for the trace log, may be reused and the associated trace stream status remains unchanged. The first trace event recorded in the trace log after the call to *posix_trace_clear()* shall be the same as the first trace event recorded in the active trace stream after the call to *posix_trace_clear()*. The *posix_log_full_status* status shall be set to *POSIX_TRACE_NOT_FULL*. There is no guarantee that all trace events that occurred during the *posix_trace_clear()* call are recorded in the trace log; the behavior with respect to trace points that may occur during this call is unspecified. If the log full policy is *POSIX_TRACE_APPEND*, the effect of a call to this function is unspecified for the trace log associated with the trace stream identified by the *trid* argument.

RETURN VALUE

Upon successful completion, the *posix_trace_clear()* function shall return a value of zero. Otherwise, it shall return the corresponding error number.

ERRORS

The *posix_trace_clear()* function shall fail if:

[EINVAL] The value of the *trid* argument does not correspond to an active trace stream.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_clear()* function may be removed in a future version.

posix_trace_clear()*System Interfaces*47946 **SEE ALSO**47947 *posix_trace_attr_destroy()*, *posix_trace_create()*, *posix_trace_get_attr()*47948 XBD *<sys/types.h>*, *<trace.h>*47949 **CHANGE HISTORY**

47950 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

47951 IEEE PASC Interpretation 1003.1 #123 is applied.

47952 **Issue 7**47953 The *posix_trace_clear()* function is marked obsolescent.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

posix_trace_close()**NAME**

posix_trace_close, posix_trace_open, posix_trace_rewind — trace log management (TRACING)

SYNOPSIS

```
#include <trace.h>

int posix_trace_close(trace_id_t trid);
int posix_trace_open(int file_desc, trace_id_t *trid);
int posix_trace_rewind(trace_id_t trid);
```

DESCRIPTION

The *posix_trace_close()* function shall deallocate the trace log identifier indicated by *trid*, and all of its associated resources. If there is no valid trace log pointed to by the *trid*, this function shall fail.

The *posix_trace_open()* function shall allocate the necessary resources and establish the connection between a trace log identified by the *file_desc* argument and a trace stream identifier identified by the object pointed to by the *trid* argument. The *file_desc* argument should be a valid open file descriptor that corresponds to a trace log. The *file_desc* argument shall be open for reading. The current trace event timestamp, which specifies the timestamp of the trace event that will be read by the next call to *posix_trace_getnext_event()*, shall be set to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

The *posix_trace_open()* function shall return a trace stream identifier in the variable pointed to by the *trid* argument, that may only be used by the following functions:

<i>posix_trace_close()</i>	<i>posix_trace_get_attr()</i>
<i>posix_trace_eventid_equal()</i>	<i>posix_trace_get_status()</i>
<i>posix_trace_eventid_get_name()</i>	<i>posix_trace_getnext_event()</i>
<i>posix_trace_eventtypelist_getnext_id()</i>	<i>posix_trace_rewind()</i>
<i>posix_trace_eventtypelist_rewind()</i>	

In particular, notice that the operations normally used by a trace controller process, such as *posix_trace_start()*, *posix_trace_stop()*, or *posix_trace_shutdown()*, cannot be invoked using the trace stream identifier returned by the *posix_trace_open()* function.

The *posix_trace_rewind()* function shall reset the current trace event timestamp, which specifies the timestamp of the trace event that will be read by the next call to *posix_trace_getnext_event()*, to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

If successful, the *posix_trace_open()* function stores the trace stream identifier value in the object pointed to by *trid*.

ERRORS

The *posix_trace_open()* function shall fail if:

[EINTR]	The operation was interrupted by a signal and thus no trace log was opened.
[EINVAL]	The object pointed to by <i>file_desc</i> does not correspond to a valid trace log.

posix_trace_close()

System Interfaces

47994 The *posix_trace_close()* and *posix_trace_rewind()* functions may fail if:

47995 [EINVAL] The object pointed to by *trid* does not correspond to a valid trace log.

47996 **EXAMPLES**

47997 None.

47998 **APPLICATION USAGE**

47999 None.

48000 **RATIONALE**

48001 None.

48002 **FUTURE DIRECTIONS**

48003 The *posix_trace_close()*, *posix_trace_open()*, and *posix_trace_rewind()* functions may be removed in
48004 a future version.

48005 **SEE ALSO**

48006 *posix_trace_get_attr()*, *posix_trace_get_filter()*, *posix_trace_getnext_event()*

48007 XBD <trace.h>

48008 **CHANGE HISTORY**

48009 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48010 IEEE PASC Interpretation 1003.1 #123 is applied.

48011 **Issue 7**

48012 The *posix_trace_close()*, *posix_trace_open()*, and *posix_trace_rewind()* functions are marked
48013 obsolescent.

System Interfaces

posix_trace_create()**NAME**

`posix_trace_create`, `posix_trace_create_withlog`, `posix_trace_flush`, `posix_trace_shutdown` — trace stream initialization, flush, and shutdown from a process (TRACING)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_create(pid_t pid,
    const trace_attr_t *restrict attr,
    trace_id_t *restrict trid);
int posix_trace_create_withlog(pid_t pid,
    const trace_attr_t *restrict attr, int file_desc,
    trace_id_t *restrict trid);
int posix_trace_flush(trace_id_t trid);
int posix_trace_shutdown(trace_id_t trid);
```

DESCRIPTION

The `posix_trace_create()` function shall create an active trace stream. It allocates all the resources needed by the trace stream being created for tracing the process specified by `pid` in accordance with the `attr` argument. The `attr` argument represents the initial attributes of the trace stream and shall have been initialized by the function `posix_trace_attr_init()` prior to the `posix_trace_create()` call. If the argument `attr` is NULL, the default attributes shall be used. The `attr` attributes object shall be manipulated through a set of functions described in the `posix_trace_attr` family of functions. If the attributes of the object pointed to by `attr` are modified later, the attributes of the trace stream shall not be affected. The *creation-time* attribute of the newly created trace stream shall be set to the value of the system clock, if the Timers option is not supported, or to the value of the CLOCK_REALTIME clock, if the Timers option is supported.

The `pid` argument represents the target process to be traced. If the process executing this function does not have appropriate privileges to trace the process identified by `pid`, an error shall be returned. If the `pid` argument is zero, the calling process shall be traced.

The `posix_trace_create()` function shall store the trace stream identifier of the new trace stream in the object pointed to by the `trid` argument. This trace stream identifier shall be used in subsequent calls to control tracing. The `trid` argument may only be used by the following functions:

<code>posix_trace_clear()</code>	<code>posix_trace_getnext_event()</code>
<code>posix_trace_eventid_equal()</code>	<code>posix_trace_shutdown()</code>
<code>posix_trace_eventid_get_name()</code>	<code>posix_trace_start()</code>
<code>posix_trace_eventtypelist_getnext_id()</code>	<code>posix_trace_stop()</code>
<code>posix_trace_eventtypelist_rewind()</code>	<code>posix_trace_timedgetnext_event()</code>
<code>posix_trace_get_attr()</code>	<code>posix_trace_trid_eventid_open()</code>
<code>posix_trace_get_status()</code>	<code>posix_trace_trygetnext_event()</code>

If the Trace Event Filter option is supported, the following additional functions may use the `trid` argument:

`posix_trace_get_filter()` `posix_trace_set_filter()`

In particular, notice that the operations normally used by a trace analyzer process, such as `posix_trace_rewind()` or `posix_trace_close()`, cannot be invoked using the trace stream identifier returned by the `posix_trace_create()` function.

posix_trace_create()*System Interfaces*

48059	TEF	A trace stream shall be created in a suspended state. If the Trace Event Filter option is supported, its trace event type filter shall be empty.
48060		
48061		The <i>posix_trace_create()</i> function may be called multiple times from the same or different processes, with the system-wide limit indicated by the runtime invariant value {TRACE_SYS_MAX}, which has the minimum value {_POSIX_TRACE_SYS_MAX}.
48062		
48063		
48064		The trace stream identifier returned by the <i>posix_trace_create()</i> function in the argument pointed to by <i>trid</i> is valid only in the process that made the function call. If it is used from another process, that is a child process, in functions defined in POSIX.1-2008, these functions shall return with the error [EINVAL].
48065		
48066		
48067		
48068	TRL	The <i>posix_trace_create_withlog()</i> function shall be equivalent to <i>posix_trace_create()</i> , except that it associates a trace log with this stream. The <i>file_desc</i> argument shall be the file descriptor designating the trace log destination. The function shall fail if this file descriptor refers to a file with a file type that is not compatible with the log policy associated with the trace log. The list of the appropriate file types that are compatible with each log policy is implementation-defined.
48069		
48070		
48071		
48072		
48073		The <i>posix_trace_create_withlog()</i> function shall return in the parameter pointed to by <i>trid</i> the trace stream identifier, which uniquely identifies the newly created trace stream, and shall be used in subsequent calls to control tracing. The <i>trid</i> argument may only be used by the following functions:
48074		
48075		
48076		
48077		<i>posix_trace_clear()</i> <i>posix_trace_get_status()</i>
48078		<i>posix_trace_eventid_equal()</i> <i>posix_trace_getnext_event()</i>
48079		<i>posix_trace_eventid_get_name()</i> <i>posix_trace_shutdown()</i>
48080		<i>posix_trace_eventtypelist_getnext_id()</i> <i>posix_trace_start()</i>
48081		<i>posix_trace_eventtypelist_rewind()</i> <i>posix_trace_stop()</i>
48082		<i>posix_trace_flush()</i> <i>posix_trace_timedgetnext_event()</i>
48083		<i>posix_trace_get_attr()</i> <i>posix_trace_trid_eventid_open()</i>
48084	TEF TRL	If the Trace Event Filter option is supported, the following additional functions may use the <i>trid</i> argument:
48085		
48086		<i>posix_trace_get_filter()</i> <i>posix_trace_set_filter()</i>
48087	TRL	In particular, notice that the operations normally used by a trace analyzer process, such as <i>posix_trace_rewind()</i> or <i>posix_trace_close()</i> , cannot be invoked using the trace stream identifier returned by the <i>posix_trace_create_withlog()</i> function.
48088		
48089		
48090		The <i>posix_trace_flush()</i> function shall initiate a flush operation which copies the contents of the trace stream identified by the argument <i>trid</i> into the trace log associated with the trace stream at the creation time. If no trace log has been associated with the trace stream pointed to by <i>trid</i> , this function shall return an error. The termination of the flush operation can be polled by the <i>posix_trace_get_status()</i> function. During the flush operation, it shall be possible to trace new trace events up to the point when the trace stream becomes full. After flushing is completed, the space used by the flushed trace events shall be available for tracing new trace events.
48091		
48092		
48093		
48094		
48095		
48096		
48097		If flushing the trace stream causes the resulting trace log to become full, the trace log full policy shall be applied. If the trace <i>log-full-policy</i> attribute is set, the following occurs:
48098		
48099		POSIX_TRACE_UNTIL_FULL
48100		The trace events that have not yet been flushed shall be discarded.

POSIX_TRACE_LOOP

The trace events that have not yet been flushed shall be written to the beginning of the trace log, overwriting previous trace events stored there.

POSIX_TRACE_APPEND

The trace events that have not yet been flushed shall be appended to the trace log.

The *posix_trace_shutdown()* function shall stop the tracing of trace events in the trace stream identified by *trid*, as if *posix_trace_stop()* had been invoked. The *posix_trace_shutdown()* function shall free all the resources associated with the trace stream.

The *posix_trace_shutdown()* function shall not return until all the resources associated with the trace stream have been freed. When the *posix_trace_shutdown()* function returns, the *trid* argument becomes an invalid trace stream identifier. A call to this function shall unconditionally deallocate the resources regardless of whether all trace events have been retrieved by the analyzer process. Any thread blocked on one of the *trace_getnext_event()* functions (which specified this *trid*) before this call is unblocked with the error [EINVAL].

If the process exits, invokes a member of the *exec* family of functions, or is terminated, the trace streams that the process had created and that have not yet been shut down, shall be automatically shut down as if an explicit call were made to the *posix_trace_shutdown()* function.

For an active trace stream with log, when the *posix_trace_shutdown()* function is called, all trace events that have not yet been flushed to the trace log shall be flushed, as in the *posix_trace_flush()* function, and the trace log shall be closed.

When a trace log is closed, all the information that may be retrieved later from the trace log through the trace interface shall have been written to the trace log. This information includes the trace attributes, the list of trace event types (with the mapping between trace event names and trace event type identifiers), and the trace status.

In addition, unspecified information shall be written to the trace log to allow detection of a valid trace log during the *posix_trace_open()* operation.

The *posix_trace_shutdown()* function shall not return until all trace events have been flushed.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

The *posix_trace_create()* and *posix_trace_create_withlog()* functions store the trace stream identifier value in the object pointed to by *trid*, if successful.

ERRORS

The *posix_trace_create()* and *posix_trace_create_withlog()* functions shall fail if:

[EAGAIN] No more trace streams can be started now. {TRACE_SYS_MAX} has been exceeded.

[EINTR] The operation was interrupted by a signal. No trace stream was created.

[EINVAL] One or more of the trace parameters specified by the *attr* parameter is invalid.

[ENOMEM] The implementation does not currently have sufficient memory to create the trace stream with the specified parameters.

[EPERM] The caller does not have appropriate privileges to trace the process specified by *pid*.

posix_trace_create()

System Interfaces

48143	[ESRCH]	The <i>pid</i> argument does not refer to an existing process.
48144	TRL	The <i>posix_trace_create_withlog()</i> function shall fail if:
48145	[EBADF]	The <i>file_desc</i> argument is not a valid file descriptor open for writing.
48146	[EINVAL]	The <i>file_desc</i> argument refers to a file with a file type that does not support the log policy associated with the trace log.
48147		
48148	[ENOSPC]	No space left on device. The device corresponding to the argument <i>file_desc</i> does not contain the space required to create this trace log.
48149		
48150	TRL	The <i>posix_trace_flush()</i> and <i>posix_trace_shutdown()</i> functions shall fail if:
48151	[EINVAL]	The value of the <i>trid</i> argument does not correspond to an active trace stream with log.
48152		
48153	[EFBIG]	The trace log file has attempted to exceed an implementation-defined maximum file size.
48154		
48155	[ENOSPC]	No space left on device.
48156	EXAMPLES	
48157	None.	
48158	APPLICATION USAGE	
48159	None.	
48160	RATIONALE	
48161	None.	
48162	FUTURE DIRECTIONS	
48163	The <i>posix_trace_create()</i> , <i>posix_trace_create_withlog()</i> , <i>posix_trace_flush()</i> , and	
48164	<i>posix_trace_shutdown()</i> functions may be removed in a future version.	
48165	SEE ALSO	
48166	<i>clock_getres()</i> , <i>exec</i> , <i>posix_trace_attr_destroy()</i> , <i>posix_trace_clear()</i> , <i>posix_trace_close()</i> ,	
48167	<i>posix_trace_eventid_equal()</i> , <i>posix_trace_eventtypelist_getnext_id()</i> , <i>posix_trace_get_attr()</i> ,	
48168	<i>posix_trace_get_filter()</i> , <i>posix_trace_getnext_event()</i> , <i>posix_trace_start()</i> , <i>posix_trace_start()</i> , <i>time()</i>	
48169	XBD <sys/types.h>, <trace.h>	
48170	CHANGE HISTORY	
48171	First released in Issue 6. Derived from IEEE Std 1003.1q-2000.	
48172	Issue 7	
48173	These functions are marked obsolescent.	
48174	SD5-XSH-ERN-154 is applied, updating the DESCRIPTION to remove the	
48175	<i>posix_trace_trygetnext_event()</i> function from the list of functions that use the <i>trid</i> argument.	

System Interfaces

posix_trace_event()**NAME**

posix_trace_event, posix_trace_eventid_open — trace functions for instrumenting application code (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

void posix_trace_event(trace_event_id_t event_id,
    const void *restrict data_ptr, size_t data_len);
int posix_trace_eventid_open(const char *restrict event_name,
    trace_event_id_t *restrict event_id);
```

DESCRIPTION

The *posix_trace_event()* function shall record the *event_id* and the user data pointed to by *data_ptr* in the trace stream into which the calling process is being traced and in which *event_id* is not filtered out. If the total size of the user trace event data represented by *data_len* is not greater than the declared maximum size for user trace event data, then the *truncation-status* attribute of the trace event recorded is POSIX_TRACE_NOT_TRUNCATED. Otherwise, the user trace event data is truncated to this declared maximum size and the *truncation-status* attribute of the trace event recorded is POSIX_TRACE_TRUNCATED_RECORD.

If there is no trace stream created for the process or if the created trace stream is not running, or if the trace event specified by *event_id* is filtered out in the trace stream, the *posix_trace_event()* function shall have no effect.

The *posix_trace_eventid_open()* function shall associate a user trace event name with a trace event type identifier for the calling process. The trace event name is the string pointed to by the argument *event_name*. It shall have a maximum of {TRACE_EVENT_NAME_MAX} characters (which has the minimum value {POSIX_TRACE_EVENT_NAME_MAX}). The number of user trace event type identifiers that can be defined for any given process is limited by the maximum value {TRACE_USER_EVENT_MAX}, which has the minimum value {POSIX_TRACE_USER_EVENT_MAX}.

If the Trace Inherit option is not supported, the *posix_trace_eventid_open()* function shall associate the user trace event name pointed to by the *event_name* argument with a trace event type identifier that is unique for the traced process, and is returned in the variable pointed to by the *event_id* argument. If the user trace event name has already been mapped for the traced process, then the previously assigned trace event type identifier shall be returned. If the per-process user trace event name limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined POSIX_TRACE_UNNAMED_USEREVENT (see Table 2-7, on page 539) user trace event shall be returned.

TRI

If the Trace Inherit option is supported, the *posix_trace_eventid_open()* function shall associate the user trace event name pointed to by the *event_name* argument with a trace event type identifier that is unique for all the processes being traced in this same trace stream, and is returned in the variable pointed to by the *event_id* argument. If the user trace event name has already been mapped for the traced processes, then the previously assigned trace event type identifier shall be returned. If the per-process user trace event name limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined POSIX_TRACE_UNNAMED_USEREVENT (Table 2-7, on page 539) user trace event shall be returned.

Note: The above procedure, together with the fact that multiple processes can only be traced into the same trace stream by inheritance, ensure that all the processes that are traced into a trace stream have the same mapping of trace event names to trace event type identifiers.

posix_trace_event()*System Interfaces*

48224 If there is no trace stream created, the *posix_trace_eventid_open()* function shall store this
 48225 information for future trace streams created for this process.

RETURN VALUE

48226 No return value is defined for the *posix_trace_event()* function.

48228 Upon successful completion, the *posix_trace_eventid_open()* function shall return a value of zero.
 48229 Otherwise, it shall return the corresponding error number. The *posix_trace_eventid_open()*
 48230 function stores the trace event type identifier value in the object pointed to by *event_id*, if
 48231 successful.

ERRORS

48232 The *posix_trace_eventid_open()* function shall fail if:

48234 [ENAMETOOLONG]

48235 The size of the name pointed to by the *event_name* argument was longer than
 48236 the implementation-defined value {TRACE_EVENT_NAME_MAX}.

EXAMPLES

48238 None.

APPLICATION USAGE

48240 None.

RATIONALE

48242 None.

FUTURE DIRECTIONS

48244 The *posix_trace_event()* and *posix_trace_eventid_open()* functions may be removed in a future
 48245 version.

SEE ALSO

48247 Table 2-7 (on page 539), *exec*, *posix_trace_eventid_equal()*, *posix_trace_start()*

48248 XBD <sys/types.h>, <trace.h>

CHANGE HISTORY

48250 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48251 IEEE PASC Interpretation 1003.1 #123 is applied.

48252 IEEE PASC Interpretation 1003.1 #127 is applied, correcting some editorial errors in the names of
 48253 the *posix_trace_eventid_open()* function and the *event_id* argument.

Issue 7

48255 The *posix_trace_event()* and *posix_trace_eventid_open()* functions are marked obsolescent.

System Interfaces

posix_trace_eventid_equal()**NAME**

`posix_trace_eventid_equal`, `posix_trace_eventid_get_name`, `posix_trace_trid_eventid_open` — manipulate the trace event type identifier (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_eventid_equal(trace_id_t trid, trace_event_id_t event1,
                             trace_event_id_t event2);
int posix_trace_eventid_get_name(trace_id_t trid,
                                trace_event_id_t event, char *event_name);
int posix_trace_trid_eventid_open(trace_id_t trid,
                                  const char *restrict event_name,
                                  trace_event_id_t *restrict event);
```

DESCRIPTION

The `posix_trace_eventid_equal()` function shall compare the trace event type identifiers *event1* and *event2* from the same trace stream or the same trace log identified by the *trid* argument. If the trace event type identifiers *event1* and *event2* are from different trace streams, the return value shall be unspecified.

The `posix_trace_eventid_get_name()` function shall return, in the argument pointed to by *event_name*, the trace event name associated with the trace event type identifier identified by the argument *event*, for the trace stream or for the trace log identified by the *trid* argument. The name of the trace event shall have a maximum of `{TRACE_EVENT_NAME_MAX}` characters (which has the minimum value `{_POSIX_TRACE_EVENT_NAME_MAX}`). Successive calls to this function with the same trace event type identifier and the same trace stream identifier shall return the same event name.

The `posix_trace_trid_eventid_open()` function shall associate a user trace event name with a trace event type identifier for a given trace stream. The trace stream is identified by the *trid* argument, and it shall be an active trace stream. The trace event name is the string pointed to by the argument *event_name*. It shall have a maximum of `{TRACE_EVENT_NAME_MAX}` characters (which has the minimum value `{_POSIX_TRACE_EVENT_NAME_MAX}`). The number of user trace event type identifiers that can be defined for any given process is limited by the maximum value `{TRACE_USER_EVENT_MAX}`, which has the minimum value `{_POSIX_TRACE_USER_EVENT_MAX}`.

If the Trace Inherit option is not supported, the `posix_trace_trid_eventid_open()` function shall associate the user trace event name pointed to by the *event_name* argument with a trace event type identifier that is unique for the process being traced in the trace stream identified by the *trid* argument, and is returned in the variable pointed to by the *event* argument. If the user trace event name has already been mapped for the traced process, then the previously assigned trace event type identifier shall be returned. If the per-process user trace event name limit represented by `{TRACE_USER_EVENT_MAX}` has been reached, the pre-defined `POSIX_TRACE_UNNAMED_USEREVENT` (see Table 2-7, on page 539) user trace event shall be returned.

If the Trace Inherit option is supported, the `posix_trace_trid_eventid_open()` function shall associate the user trace event name pointed to by the *event_name* argument with a trace event type identifier that is unique for all the processes being traced in the trace stream identified by the *trid* argument, and is returned in the variable pointed to by the *event* argument. If the user trace event name has already been mapped for the traced processes, then the previously assigned trace event type identifier shall be returned. If the per-process user trace event name limit represented by `{TRACE_USER_EVENT_MAX}` has been reached, the pre-defined

posix_trace_eventid_equal()

System Interfaces

48304 POSIX_TRACE_UNNAMED_USEREVENT (see Table 2-7, on page 539) user trace event shall be
 48305 returned.

RETURN VALUE

48307 TEF Upon successful completion, the *posix_trace_eventid_get_name()* and
 48308 *posix_trace_trid_eventid_open()* functions shall return a value of zero. Otherwise, they shall return
 48309 the corresponding error number.

48310 The *posix_trace_eventid_equal()* function shall return a non-zero value if *event1* and *event2* are
 48311 equal; otherwise, a value of zero shall be returned. No errors are defined. If either *event1* or
 48312 *event2* are not valid trace event type identifiers for the trace stream specified by *trid* or if the *trid*
 48313 is invalid, the behavior shall be unspecified.

48314 The *posix_trace_eventid_get_name()* function stores the trace event name value in the object
 48315 pointed to by *event_name*, if successful.

48316 TEF The *posix_trace_trid_eventid_open()* function stores the trace event type identifier value in the
 48317 object pointed to by *event*, if successful.

ERRORS

48319 TEF The *posix_trace_eventid_get_name()* and *posix_trace_trid_eventid_open()* functions shall fail if:

48320 [EINVAL] The *trid* argument was not a valid trace stream identifier.

48321 TEF The *posix_trace_trid_eventid_open()* function shall fail if:

48322 TEF [ENAMETOOLONG]

48323 The size of the name pointed to by the *event_name* argument was longer than
 48324 the implementation-defined value [TRACE_EVENT_NAME_MAX].

48325 The *posix_trace_eventid_get_name()* function shall fail if:

48326 [EINVAL] The trace event type identifier *event* was not associated with any name.

EXAMPLES

48328 None.

APPLICATION USAGE

48330 None.

RATIONALE

48332 None.

FUTURE DIRECTIONS

48334 The *posix_trace_eventid_equal()*, *posix_trace_eventid_get_name()*, and
 48335 *posix_trace_trid_eventid_open()* functions may be removed in a future version.

SEE ALSO

48337 Table 2-7 (on page 539), *exec*, *posix_trace_event()*, *posix_trace_getnext_event()*

48338 XBD <trace.h>

CHANGE HISTORY

48340 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48341 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

Issue 7

48343 These functions are marked obsolescent.

*System Interfaces***posix_trace_eventid_open()**48344 **NAME**48345 **posix_trace_eventid_open** — trace functions for instrumenting application code (**TRACING**)48346 **SYNOPSIS**

```
48347 OB TRC #include <sys/types.h>  
48348          #include <trace.h>  
  
48349          int posix_trace_eventid_open(const char *restrict event_name,  
48350                                     trace_event_id_t *restrict event_id);
```

48351 **DESCRIPTION**48352 Refer to *posix_trace_event()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_eventset_add()

System Interfaces

48353 **NAME**

48354 `posix_trace_eventset_add`, `posix_trace_eventset_del`, `posix_trace_eventset_empty`,
 48355 `posix_trace_eventset_fill`, `posix_trace_eventset_ismember` — manipulate trace event type sets
 48356 (TRACING)

48357 **SYNOPSIS**

```
48358 OB TRC #include <trace.h>
48359 TEF int posix_trace_eventset_add(trace_event_id_t event_id,
48360 trace_event_set_t *set);
48361 int posix_trace_eventset_del(trace_event_id_t event_id,
48362 trace_event_set_t *set);
48363 int posix_trace_eventset_empty(trace_event_set_t *set);
48364 int posix_trace_eventset_fill(trace_event_set_t *set, int what);
48365 int posix_trace_eventset_ismember(trace_event_id_t event_id,
48366 const trace_event_set_t *restrict set, int *restrict ismember);
```

48367 **DESCRIPTION**

48368 These primitives manipulate sets of trace event types. They operate on data objects addressable
 48369 by the application, not on the current trace event filter of any trace stream.

48370 The `posix_trace_eventset_add()` and `posix_trace_eventset_del()` functions, respectively, shall add or
 48371 delete the individual trace event type specified by the value of the argument `event_id` to or from
 48372 the trace event type set pointed to by the argument `set`. Adding a trace event type already in the
 48373 set or deleting a trace event type not in the set shall not be considered an error.

48374 The `posix_trace_eventset_empty()` function shall initialize the trace event type set pointed to by
 48375 the `set` argument such that all trace event types defined, both system and user, shall be excluded
 48376 from the set.

48377 The `posix_trace_eventset_fill()` function shall initialize the trace event type set pointed to by the
 48378 argument `set`, such that the set of trace event types defined by the argument `what` shall be
 48379 included in the set. The value of the argument `what` shall consist of one of the following values,
 48380 as defined in the `<trace.h>` header:

48381 **POSIX_TRACE_WOPIID_EVENTS**

48382 All the process-independent implementation-defined system trace event types are included
 48383 in the set.

48384 **POSIX_TRACE_SYSTEM_EVENTS**

48385 All the implementation-defined system trace event types are included in the set, as are those
 48386 defined in POSIX.1-2008.

48387 **POSIX_TRACE_ALL_EVENTS**

48388 All trace event types defined, both system and user, are included in the set.

48389 Applications shall call either `posix_trace_eventset_empty()` or `posix_trace_eventset_fill()` at least
 48390 once for each object of type `trace_event_set_t` prior to any other use of that object. If such an
 48391 object is not initialized in this way, but is nonetheless supplied as an argument to any of the
 48392 `posix_trace_eventset_add()`, `posix_trace_eventset_del()`, or `posix_trace_eventset_ismember()` functions,
 48393 the results are undefined.

48394 The `posix_trace_eventset_ismember()` function shall test whether the trace event type specified by
 48395 the value of the argument `event_id` is a member of the set pointed to by the argument `set`. The
 48396 value returned in the object pointed to by `ismember` argument is zero if the trace event type
 48397 identifier is not a member of the set and a value different from zero if it is a member of the set.

48398 RETURN VALUE

48399 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 48400 return the corresponding error number.

48401 ERRORS

48402 These functions may fail if:

48403 [EINVAL] The value of one of the arguments is invalid.

48404 EXAMPLES

48405 None.

48406 APPLICATION USAGE

48407 None.

48408 RATIONALE

48409 None.

48410 FUTURE DIRECTIONS

48411 The *posix_trace_eventset_add()*, *posix_trace_eventset_del()*, *posix_trace_eventset_empty()*,
 48412 *posix_trace_eventset_fill()*, and *posix_trace_eventset_ismember()* functions may be removed in a
 48413 future version.

48414 SEE ALSO

48415 *posix_trace_eventid_equal()*, *posix_trace_get_filter()*

48416 XBD <trace.h>

48417 CHANGE HISTORY

48418 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48419 Issue 7

48420 The *posix_trace_eventset_add()*, *posix_trace_eventset_del()*, *posix_trace_eventset_empty()*,
 48421 *posix_trace_eventset_fill()*, and *posix_trace_eventset_ismember()* functions are marked obsolescent.

posix_trace_eventtypelist_getnext_id()

System Interfaces

48422 **NAME**

48423 `posix_trace_eventtypelist_getnext_id`, `posix_trace_eventtypelist_rewind` — iterate over a
 48424 mapping of trace event types (**TRACING**)

48425 **SYNOPSIS**

```
48426 OB TRC #include <trace.h>

48427 int posix_trace_eventtypelist_getnext_id(trace_id_t trid,
48428     trace_event_id_t *restrict event, int *restrict unavailable);
48429 int posix_trace_eventtypelist_rewind(trace_id_t trid);
```

48430 **DESCRIPTION**

48431 The first time `posix_trace_eventtypelist_getnext_id()` is called, the function shall return in the
 48432 variable pointed to by *event* the first trace event type identifier of the list of trace events of the
 48433 trace stream identified by the *trid* argument. Successive calls to
 48434 `posix_trace_eventtypelist_getnext_id()` return in the variable pointed to by *event* the next trace
 48435 event type identifier in that same list. Each time a trace event type identifier is successfully
 48436 written into the variable pointed to by the *event* argument, the variable pointed to by the
 48437 *unavailable* argument shall be set to zero. When no more trace event type identifiers are available,
 48438 and so none is returned, the variable pointed to by the *unavailable* argument shall be set to a
 48439 value different from zero.

48440 The `posix_trace_eventtypelist_rewind()` function shall reset the next trace event type identifier to
 48441 be read to the first trace event type identifier from the list of trace events used in the trace stream
 48442 identified by *trid*.

48443 **RETURN VALUE**

48444 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 48445 return the corresponding error number.

48446 The `posix_trace_eventtypelist_getnext_id()` function stores the trace event type identifier value in
 48447 the object pointed to by *event*, if successful.

48448 **ERRORS**

48449 These functions shall fail if:

48450 [EINVAL] The *trid* argument was not a valid trace stream identifier.

48451 **EXAMPLES**

48452 None.

48453 **APPLICATION USAGE**

48454 None.

48455 **RATIONALE**

48456 None.

48457 **FUTURE DIRECTIONS**

48458 The `posix_trace_eventtypelist_getnext_id()` and `posix_trace_eventtypelist_rewind()` functions may be
 48459 removed in a future version.

48460 **SEE ALSO**

48461 [*posix_trace_event\(\)*](#), [*posix_trace_eventid_equal\(\)*](#), [*posix_trace_getnext_event\(\)*](#)

48462 XBD [*<trace.h>*](#)

CHANGE HISTORY

48463 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48464 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

Issue 7

48466 The *posix_trace_eventtypelist_getnext_id()* and *posix_trace_eventtypelist_rewind()* functions are
48467 marked obsolescent.
48468

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_flush()*System Interfaces*48469 **NAME**

48470 posix_trace_flush — trace stream flush from a process (TRACING)

48471 **SYNOPSIS**

```
48472 OB TRC  #include <sys/types.h>  
48473          #include <trace.h>  
48474 TRL      int posix_trace_flush(trace_id_t trid);
```

48475 **DESCRIPTION**48476 Refer to *posix_trace_create()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

posix_trace_get_attr()**NAME**

posix_trace_get_attr, posix_trace_get_status — retrieve the trace attributes or trace status (TRACING)

SYNOPSIS

```
#include <trace.h>
```

```
int posix_trace_get_attr(trace_id_t trid, trace_attr_t *attr);
int posix_trace_get_status(trace_id_t trid,
    struct posix_trace_status_info *statusinfo);
```

DESCRIPTION

The *posix_trace_get_attr()* function shall copy the attributes of the active trace stream identified by *trid* into the object pointed to by the *attr* argument. If the Trace Log option is supported, *trid* may represent a pre-recorded trace log.

The *posix_trace_get_status()* function shall return, in the structure pointed to by the *statusinfo* argument, the current trace status for the trace stream identified by the *trid* argument. These status values returned in the structure pointed to by *statusinfo* shall have been appropriately read to ensure that the returned values are consistent. If the Trace Log option is supported and the *trid* argument refers to a pre-recorded trace stream, the status shall be the status of the completed trace stream.

Each time the *posix_trace_get_status()* function is used, the overrun status of the trace stream shall be reset to POSIX_TRACE_NO_OVERRUN immediately after the call completes. If the Trace Log option is supported, the *posix_trace_get_status()* function shall behave the same as when the option is not supported except for the following differences:

- If the *trid* argument refers to a trace stream with log, each time the *posix_trace_get_status()* function is used, the log overrun status of the trace stream shall be reset to POSIX_TRACE_NO_OVERRUN and the *flush_error* status shall be reset to zero immediately after the call completes.
- If the *trid* argument refers to a pre-recorded trace stream, the status returned shall be the status of the completed trace stream and the status values of the trace stream shall not be reset.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

The *posix_trace_get_attr()* function stores the trace attributes in the object pointed to by *attr*, if successful.

The *posix_trace_get_status()* function stores the trace status in the object pointed to by *statusinfo*, if successful.

ERRORS

These functions shall fail if:

[EINVAL] The trace stream argument *trid* does not correspond to a valid active trace stream or a valid trace log.

posix_trace_get_attr()

System Interfaces

48517 **EXAMPLES**

48518 None.

48519 **APPLICATION USAGE**

48520 None.

48521 **RATIONALE**

48522 None.

48523 **FUTURE DIRECTIONS**48524 The *posix_trace_get_attr()* and *posix_trace_get_status()* functions may be removed in a future
48525 version.48526 **SEE ALSO**48527 *posix_trace_attr_destroy()*, *posix_trace_close()*, *posix_trace_create()*

48528 XBD <trace.h>

48529 **CHANGE HISTORY**

48530 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48531 IEEE PASC Interpretation 1003.1 #123 is applied.

48532 **Issue 7**48533 The *posix_trace_get_attr()* and *posix_trace_get_status()* functions are marked obsolescent.

System Interfaces

posix_trace_get_filter()**NAME**

posix_trace_get_filter, posix_trace_set_filter — retrieve and set the filter of an initialized trace stream (TRACING)

SYNOPSIS

OB TRC `#include <trace.h>`

TEF `int posix_trace_get_filter(trace_id_t trid, trace_event_set_t *set);`
`int posix_trace_set_filter(trace_id_t trid,`
`const trace_event_set_t *set, int how);`

DESCRIPTION

The *posix_trace_get_filter()* function shall retrieve, into the argument pointed to by *set*, the actual trace event filter from the trace stream specified by *trid*.

The *posix_trace_set_filter()* function shall change the set of filtered trace event types after a trace stream identified by the *trid* argument is created. This function may be called prior to starting the trace stream, or while the trace stream is active. By default, if no call is made to *posix_trace_set_filter()*, all trace events shall be recorded (that is, none of the trace event types are filtered out).

If this function is called while the trace is in progress, a special system trace event, `POSIX_TRACE_FILTER`, shall be recorded in the trace indicating both the old and the new sets of filtered trace event types (see Table 2-4 (on page 537) and Table 2-6, on page 538).

If the *posix_trace_set_filter()* function is interrupted by a signal, an error shall be returned and the filter shall not be changed. In this case, the state of the trace stream shall not be changed.

The value of the argument *how* indicates the manner in which the set is to be changed and shall have one of the following values, as defined in the `<trace.h>` header:

`POSIX_TRACE_SET_EVENTSET`

The resulting set of trace event types to be filtered shall be the trace event type set pointed to by the argument *set*.

`POSIX_TRACE_ADD_EVENTSET`

The resulting set of trace event types to be filtered shall be the union of the current set and the trace event type set pointed to by the argument *set*.

`POSIX_TRACE_SUB_EVENTSET`

The resulting set of trace event types to be filtered shall be all trace event types in the current set that are not in the set pointed to by the argument *set*; that is, remove each element of the specified set from the current filter.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

The *posix_trace_get_filter()* function stores the set of filtered trace event types in *set*, if successful.

ERRORS

These functions shall fail if:

[EINVAL] The value of the *trid* argument does not correspond to an active trace stream or the value of the argument pointed to by *set* is invalid.

[EINTR] The operation was interrupted by a signal.

posix_trace_get_filter()

System Interfaces

48576 **EXAMPLES**

48577 None.

48578 **APPLICATION USAGE**

48579 None.

48580 **RATIONALE**

48581 None.

48582 **FUTURE DIRECTIONS**48583 The *posix_trace_get_filter()* and *posix_trace_set_filter()* functions may be removed in a future
48584 version.48585 **SEE ALSO**48586 [Table 2-4](#) (on page 537), [Table 2-6](#) (on page 538), [posix_trace_eventset_add\(\)](#)48587 XBD [<trace.h>](#)48588 **CHANGE HISTORY**

48589 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48590 IEEE PASC Interpretation 1003.1 #123 is applied.

48591 **Issue 7**48592 The *posix_trace_get_filter()* and *posix_trace_set_filter()* functions are marked obsolescent.

*System Interfaces***posix_trace_get_status()**48593 **NAME**

48594 posix_trace_get_status — retrieve the trace status (TRACING)

48595 **SYNOPSIS**

```
48596 OB TRC #include <trace.h>
48597         int posix_trace_get_status(trace_id_t trid,
48598         struct posix_trace_status_info *statusinfo);
```

48599 **DESCRIPTION**48600 Refer to *posix_trace_get_attr()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_getnext_event()

System Interfaces

48601 **NAME**

48602 `posix_trace_getnext_event`, `posix_trace_timedgetnext_event`, `posix_trace_trygetnext_event` —
 48603 retrieve a trace event (**TRACING**)

48604 **SYNOPSIS**

```
48605 OB TRC #include <sys/types.h>
48606          #include <trace.h>

48607 int posix_trace_getnext_event(trace_id_t trid,
48608                             struct posix_trace_event_info *restrict event,
48609                             void *restrict data, size_t num_bytes,
48610                             size_t *restrict data_len, int *restrict unavailable);
48611 int posix_trace_timedgetnext_event(trace_id_t trid,
48612                                   struct posix_trace_event_info *restrict event,
48613                                   void *restrict data, size_t num_bytes,
48614                                   size_t *restrict data_len, int *restrict unavailable,
48615                                   const struct timespec *restrict abstime);
48616 int posix_trace_trygetnext_event(trace_id_t trid,
48617                                 struct posix_trace_event_info *restrict event,
48618                                 void *restrict data, size_t num_bytes,
48619                                 size_t *restrict data_len, int *restrict unavailable);
```

48620 **DESCRIPTION**

48621 The `posix_trace_getnext_event()` function shall report a recorded trace event either from an active
 48622 TRL trace stream without log or a pre-recorded trace stream identified by the *trid* argument. The
 48623 `posix_trace_trygetnext_event()` function shall report a recorded trace event from an active trace
 48624 stream without log identified by the *trid* argument.

48625 The trace event information associated with the recorded trace event shall be copied by the
 48626 function into the structure pointed to by the argument *event* and the data associated with the
 48627 trace event shall be copied into the buffer pointed to by the *data* argument.

48628 The `posix_trace_getnext_event()` function shall block if the *trid* argument identifies an active trace
 48629 stream and there is currently no trace event ready to be retrieved. When returning, if a recorded
 48630 trace event was reported, the variable pointed to by the *unavailable* argument shall be set to zero.
 48631 Otherwise, the variable pointed to by the *unavailable* argument shall be set to a value different
 48632 from zero.

48633 The `posix_trace_timedgetnext_event()` function shall attempt to get another trace event from an
 48634 active trace stream without log, as in the `posix_trace_getnext_event()` function. However, if no
 48635 trace event is available from the trace stream, the implied wait shall be terminated when the
 48636 timeout specified by the argument *abstime* expires, and the function shall return the error
 48637 [ETIMEDOUT].

48638 The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the
 48639 clock upon which timeouts are based (that is, when the value of that clock equals or exceeds
 48640 *abstime*), or if the absolute time specified by *abstime* has already passed at the time of the call.

48641 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
 48642 be the resolution of the clock on which it is based. The **timespec** data type is defined in the
 48643 **<time.h>** header.

48644 Under no circumstance shall the function fail with a timeout if a trace event is immediately
 48645 available from the trace stream. The validity of the *abstime* argument need not be checked if a
 48646 trace event is immediately available from the trace stream.

The behavior of this function for a pre-recorded trace stream is unspecified.

TRL The *posix_trace_trygetnext_event()* function shall not block. This function shall return an error if the *trid* argument identifies a pre-recorded trace stream. If a recorded trace event was reported, the variable pointed to by the *unavailable* argument shall be set to zero. Otherwise, if no trace event was reported, the variable pointed to by the *unavailable* argument shall be set to a value different from zero.

The argument *num_bytes* shall be the size of the buffer pointed to by the *data* argument. The argument *data_len* reports to the application the length in bytes of the data record just transferred. If *num_bytes* is greater than or equal to the size of the data associated with the trace event pointed to by the *event* argument, all the recorded data shall be transferred. In this case, the *truncation-status* member of the trace event structure shall be either `POSIX_TRACE_NOT_TRUNCATED`, if the trace event data was recorded without truncation while tracing, or `POSIX_TRACE_TRUNCATED_RECORD`, if the trace event data was truncated when it was recorded. If the *num_bytes* argument is less than the length of recorded trace event data, the data transferred shall be truncated to a length of *num_bytes*, the value stored in the variable pointed to by *data_len* shall be equal to *num_bytes*, and the *truncation-status* member of the *event* structure argument shall be set to `POSIX_TRACE_TRUNCATED_READ` (see the **posix_trace_event_info** structure defined in `<trace.h>`).

The report of a trace event shall be sequential starting from the oldest recorded trace event. Trace events shall be reported in the order in which they were generated, up to an implementation-defined time resolution that causes the ordering of trace events occurring very close to each other to be unknown. Once reported, a trace event cannot be reported again from an active trace stream. Once a trace event is reported from an active trace stream without log, the trace stream shall make the resources associated with that trace event available to record future generated trace events.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

If successful, these functions store:

- The recorded trace event in the object pointed to by *event*
- The trace event information associated with the recorded trace event in the object pointed to by *data*
- The length of this trace event information in the object pointed to by *data_len*
- The value of zero in the object pointed to by *unavailable*

ERRORS

These functions shall fail if:

[EINVAL] The trace stream identifier argument *trid* is invalid.

The *posix_trace_getnext_event()* and *posix_trace_timedgetnext_event()* functions shall fail if:

[EINTR] The operation was interrupted by a signal, and so the call had no effect.

The *posix_trace_trygetnext_event()* function shall fail if:

[EINVAL] The trace stream identifier argument *trid* does not correspond to an active trace stream.

posix_trace_getnext_event()*System Interfaces*

48689 The *posix_trace_timedgetnext_event()* function shall fail if:

48690 [EINVAL] There is no trace event immediately available from the trace stream, and the
48691 *timeout* argument is invalid.

48692 [ETIMEDOUT] No trace event was available from the trace stream before the specified
48693 *timeout* expired.

48694 **EXAMPLES**

48695 None.

48696 **APPLICATION USAGE**

48697 None.

48698 **RATIONALE**

48699 None.

48700 **FUTURE DIRECTIONS**

48701 These functions may be removed in a future version.

48702 **SEE ALSO**

48703 *posix_trace_close()*, *posix_trace_create()*

48704 XBD <sys/types.h>, <trace.h>

48705 **CHANGE HISTORY**

48706 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

48707 IEEE PASC Interpretation 1003.1 #123 is applied.

48708 **Issue 7**

48709 The *posix_trace_getnext_event()*, *posix_trace_timedgetnext_event()*, and
48710 *posix_trace_trygetnext_event()* functions are marked obsolescent.

48711 Functionality relating to the Timers option is moved to the Base.

*System Interfaces***posix_trace_open()**48712 **NAME**

48713 posix_trace_open, posix_trace_rewind — trace log management (TRACING)

48714 **SYNOPSIS**48715 OB TRC `#include <trace.h>`48716 TRL `int posix_trace_open(int file_desc, trace_id_t *trid);`48717 `int posix_trace_rewind(trace_id_t trid);`48718 **DESCRIPTION**48719 Refer to *posix_trace_close()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_set_filter()*System Interfaces*48720 **NAME**48721 `posix_trace_set_filter` — set filter of an initialized trace stream (**TRACING**)48722 **SYNOPSIS**48723 OB TRC `#include <trace.h>`48724 TEF `int posix_trace_set_filter(trace_id_t trid,`
48725 `const trace_event_set_t *set, int how);`48726 **DESCRIPTION**48727 Refer to *posix_trace_get_filter()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***posix_trace_shutdown()**48728 **NAME**48729 posix_trace_shutdown — trace stream shutdown from a process (**TRACING**)48730 **SYNOPSIS**

```
48731 OB TRC #include <sys/types.h>  
48732         #include <trace.h>  
48733         int posix_trace_shutdown(trace_id_t trid);
```

48734 **DESCRIPTION**48735 Refer to *posix_trace_create()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_start()

System Interfaces

48736 **NAME**48737 `posix_trace_start`, `posix_trace_stop` — trace start and stop (**TRACING**)48738 **SYNOPSIS**

```
48739 OB TRC #include <trace.h>
48740
48740 int posix_trace_start(trace_id_t trid);
48741 int posix_trace_stop (trace_id_t trid);
```

48742 **DESCRIPTION**

48743 The `posix_trace_start()` and `posix_trace_stop()` functions, respectively, shall start and stop the trace
 48744 stream identified by the argument *trid*.

48745 The effect of calling the `posix_trace_start()` function shall be recorded in the trace stream as the
 48746 POSIX_TRACE_START system trace event and the status of the trace stream shall become
 48747 POSIX_TRACE_RUNNING. If the trace stream is in progress when this function is called, the
 48748 POSIX_TRACE_START system trace event shall not be recorded and the trace stream shall
 48749 continue to run. If the trace stream is full, the POSIX_TRACE_START system trace event shall
 48750 not be recorded and the status of the trace stream shall not be changed.

48751 The effect of calling the `posix_trace_stop()` function shall be recorded in the trace stream as the
 48752 POSIX_TRACE_STOP system trace event and the status of the trace stream shall become
 48753 POSIX_TRACE_SUSPENDED. If the trace stream is suspended when this function is called, the
 48754 POSIX_TRACE_STOP system trace event shall not be recorded and the trace stream shall remain
 48755 suspended. If the trace stream is full, the POSIX_TRACE_STOP system trace event shall not be
 48756 recorded and the status of the trace stream shall not be changed.

48757 **RETURN VALUE**

48758 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 48759 return the corresponding error number.

48760 **ERRORS**

48761 These functions shall fail if:

48762 [EINVAL] The value of the argument *trid* does not correspond to an active trace stream
 48763 and thus no trace stream was started or stopped.

48764 [EINTR] The operation was interrupted by a signal and thus the trace stream was not
 48765 necessarily started or stopped.

48766 **EXAMPLES**

48767 None.

48768 **APPLICATION USAGE**

48769 None.

48770 **RATIONALE**

48771 None.

48772 **FUTURE DIRECTIONS**

48773 The `posix_trace_start()` and `posix_trace_stop()` functions may be removed in a future version.

48774 **SEE ALSO**

48775 [*posix_trace_create\(\)*](#)

48776 XBD [*<trace.h>*](#)

CHANGE HISTORY

- 48777 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.
- 48778 IEEE PASC Interpretation 1003.1 #123 is applied.
- 48779
- 48780 **Issue 7**
- 48781 The *posix_trace_start()* and *posix_trace_stop()* functions are marked obsolescent.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_timedgetnext_event()*System Interfaces*48782 **NAME**48783 **posix_trace_timedgetnext_event** — retrieve a trace event (TRACING)48784 **SYNOPSIS**

```
48785 OB TRC #include <sys/types.h>
48786         #include <trace.h>
48787
48787         int posix_trace_timedgetnext_event(trace_id_t trid,
48788         struct posix_trace_event_info *restrict event,
48789         void *restrict data, size_t num_bytes,
48790         size_t *restrict data_len, int *restrict unavailable,
48791         const struct timespec *restrict abstime);
```

48792 **DESCRIPTION**48793 Refer to *posix_trace_getnext_event()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***posix_trace_trid_eventid_open()**48794 **NAME**

48795 posix_trace_trid_eventid_open — open a trace event type identifier (TRACING)

48796 **SYNOPSIS**

```
48797 OB TRC  #include <trace.h>
48798 TEF      int posix_trace_trid_eventid_open(trace_id_t trid,
48799      const char *restrict event_name,
48800      trace_event_id_t *restrict event);
```

48801 **DESCRIPTION**48802 Refer to *posix_trace_eventid_equal()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

posix_trace_trygetnext_event()*System Interfaces*48803 **NAME**48804 **posix_trace_trygetnext_event** — retrieve a trace event (**TRACING**)48805 **SYNOPSIS**

```
48806 OB TRC #include <sys/types.h>
48807         #include <trace.h>
48808
48808         int posix_trace_trygetnext_event(trace_id_t trid,
48809         struct posix_trace_event_info *restrict event,
48810         void *restrict data, size_t num_bytes,
48811         size_t *restrict data_len, int *restrict unavailable);
```

48812 **DESCRIPTION**48813 Refer to [posix_trace_getnext_event\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

posix_typed_mem_get_info — query typed memory information (ADVANCED REALTIME)

SYNOPSIS

```
#include <sys/mman.h>

int posix_typed_mem_get_info(int fildes,
    struct posix_typed_mem_info *info);
```

DESCRIPTION

The *posix_typed_mem_get_info()* function shall return, in the *posix_tmi_length* field of the **posix_typed_mem_info** structure pointed to by *info*, the maximum length which may be successfully allocated by the typed memory object designated by *fildes*. This maximum length shall take into account the flag `POSIX_TYPED_MEM_ALLOCATE` or `POSIX_TYPED_MEM_ALLOCATE_CONTIG` specified when the typed memory object represented by *fildes* was opened. The maximum length is dynamic; therefore, the value returned is valid only while the current mapping of the corresponding typed memory pool remains unchanged.

If *fildes* represents a typed memory object opened with neither the `POSIX_TYPED_MEM_ALLOCATE` flag nor the `POSIX_TYPED_MEM_ALLOCATE_CONTIG` flag specified, the returned value of *info->posix_tmi_length* is unspecified.

The *posix_typed_mem_get_info()* function may return additional implementation-defined information in other fields of the **posix_typed_mem_info** structure pointed to by *info*.

If the memory object specified by *fildes* is not a typed memory object, then the behavior of this function is undefined.

RETURN VALUE

Upon successful completion, the *posix_typed_mem_get_info()* function shall return zero; otherwise, the corresponding error status value shall be returned.

ERRORS

The *posix_typed_mem_get_info()* function shall fail if:

[EBADF] The *fildes* argument is not a valid open file descriptor.

[ENODEV] The *fildes* argument is not connected to a memory object supported by this function.

This function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

An application that needs to allocate a block of typed memory with length dependent upon the amount of memory currently available must either query the typed memory object to obtain the amount available, or repeatedly invoke *mmap()* attempting to guess an appropriate length. While the latter method is existing practice with *malloc()*, it is awkward and imprecise. The *posix_typed_mem_get_info()* function allows an application to immediately determine available memory. This is particularly important for typed memory objects that may in some cases be scarce resources. Note that when a typed memory pool is a shared resource, some form of mutual-exclusion or synchronization may be required while typed memory is being queried and

posix_typed_mem_get_info()*System Interfaces*

48858 allocated to prevent race conditions.

48859 The existing *fstat()* function is not suitable for this purpose. We realize that implementations
48860 may wish to provide other attributes of typed memory objects (for example, alignment
48861 requirements, page size, and so on). The *fstat()* function returns a structure which is not
48862 extensible and, furthermore, contains substantial information that is inappropriate for typed
48863 memory objects.

48864 FUTURE DIRECTIONS

48865 None.

48866 SEE ALSO

48867 *fstat()*, *mmap()*, *posix_typed_mem_open()*

48868 XBD <sys/mman.h>

48869 CHANGE HISTORY

48870 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

posix_typed_mem_open — open a typed memory object (ADVANCED REALTIME)

SYNOPSIS

```
#include <sys/mman.h>

int posix_typed_mem_open(const char *name, int oflag, int tflag);
```

DESCRIPTION

The *posix_typed_mem_open()* function shall establish a connection between the typed memory object specified by the string pointed to by *name* and a file descriptor. It shall create an open file description that refers to the typed memory object and a file descriptor that refers to that open file description. The file descriptor is used by other functions to refer to that typed memory object. It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The *name* argument conforms to the construction rules for a pathname, except that the interpretation of <slash> characters other than the leading <slash> character in *name* is implementation-defined, and that the length limits for the *name* argument are implementation-defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If *name* begins with the <slash> character, then processes calling *posix_typed_mem_open()* with the same value of *name* shall refer to the same typed memory object. If *name* does not begin with the <slash> character, the effect is implementation-defined.

Each typed memory object supported in a system shall be identified by a name which specifies not only its associated typed memory pool, but also the path or port by which it is accessed. That is, the same typed memory pool accessed via several different ports shall have several different corresponding names. The binding between names and typed memory objects is established in an implementation-defined manner. Unlike shared memory objects, there is no way within POSIX.1-2008 for a program to create a typed memory object.

The value of *tflag* shall determine how the typed memory object behaves when subsequently mapped by calls to *mmap()*. At most, one of the following flags defined in <sys/mman.h> may be specified:

POSIX_TYPED_MEM_ALLOCATE

Allocate on *mmap()*.

POSIX_TYPED_MEM_ALLOCATE_CONTIG

Allocate contiguously on *mmap()*.

POSIX_TYPED_MEM_MAP_ALLOCATABLE

Map on *mmap()*, without affecting allocatability.

If *tflag* has the flag POSIX_TYPED_MEM_ALLOCATE specified, any subsequent call to *mmap()* using the returned file descriptor shall result in allocation and mapping of typed memory from the specified typed memory pool. The allocated memory may be a contiguous previously unallocated area of the typed memory pool or several non-contiguous previously unallocated areas (mapped to a contiguous portion of the process address space). If *tflag* has the flag POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap()* using the returned file descriptor shall result in allocation and mapping of a single contiguous previously unallocated area of the typed memory pool (also mapped to a contiguous portion of the process address space). If *tflag* has none of the flags POSIX_TYPED_MEM_ALLOCATE or POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap()* using the returned file descriptor shall map an application-chosen area from the specified typed memory pool such that this mapped area becomes unavailable for allocation until unmapped by all processes. If *tflag* has the flag POSIX_TYPED_MEM_MAP_ALLOCATABLE specified, any

posix_typed_mem_open()

System Interfaces

subsequent call to *mmap()* using the returned file descriptor shall map an application-chosen area from the specified typed memory pool without an effect on the availability of that area for allocation; that is, mapping such an object leaves each byte of the mapped area unallocated if it was unallocated prior to the mapping or allocated if it was allocated prior to the mapping. Appropriate privileges to specify the `POSIX_TYPED_MEM_MAP_ALLOCATABLE` flag are implementation-defined.

If successful, *posix_typed_mem_open()* shall return a file descriptor for the typed memory object that is the lowest numbered file descriptor not currently open for that process. The open file description is new, and therefore the file descriptor shall not share it with any other processes. It is unspecified whether the file offset is set. The `FD_CLOEXEC` file descriptor flag associated with the new file descriptor shall be cleared.

The behavior of *msync()*, *ftruncate()*, and all file operations other than *mmap()*, *posix_mem_offset()*, *posix_typed_mem_get_info()*, *fstat()*, *dup()*, *dup2()*, and *close()*, is unspecified when passed a file descriptor connected to a typed memory object by this function.

The file status flags of the open file description shall be set according to the value of *oflag*. Applications shall specify exactly one of the three access mode values described below and defined in the `<fcntl.h>` header, as the value of *oflag*.

48935	<code>O_RDONLY</code>	Open for read access only.
48936	<code>O_WRONLY</code>	Open for write access only.
48937	<code>O_RDWR</code>	Open for read or write access.

RETURN VALUE

Upon successful completion, the *posix_typed_mem_open()* function shall return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, it shall return `-1` and set *errno* to indicate the error.

ERRORS

The *posix_typed_mem_open()* function shall fail if:

48944	<code>[EACCES]</code>	The typed memory object exists and the permissions specified by <i>oflag</i> are denied.
48946	<code>[EINTR]</code>	The <i>posix_typed_mem_open()</i> operation was interrupted by a signal.
48947	<code>[EINVAL]</code>	The flags specified in <i>tflag</i> are invalid (more than one of <code>POSIX_TYPED_MEM_ALLOCATE</code> , <code>POSIX_TYPED_MEM_ALLOCATE_CONTIG</code> , or <code>POSIX_TYPED_MEM_MAP_ALLOCATABLE</code> is specified).
48951	<code>[EMFILE]</code>	All file descriptors available to the process are currently open.
48952	<code>[ENAMETOOLONG]</code>	The length of the <i>name</i> argument exceeds <code>{_POSIX_PATH_MAX}</code> on systems that do not support the XSI option or exceeds <code>{_XOPEN_PATH_MAX}</code> on XSI systems, or has a pathname component that is longer than <code>{_POSIX_NAME_MAX}</code> on systems that do not support the XSI option or longer than <code>{_XOPEN_NAME_MAX}</code> on XSI systems.
48954 XSI		
48955		
48956 XSI		
48957		
48958	<code>[ENFILE]</code>	Too many file descriptors are currently open in the system.
48959	<code>[ENOENT]</code>	The named typed memory object does not exist.

48960 [EPERM] The caller lacks appropriate privileges to specify the
 48961 POSIX_TYPED_MEM_MAP_ALLOCATABLE flag in the *tflag* argument.

EXAMPLES

48962 None.
 48963

APPLICATION USAGE

48964 None.
 48965

RATIONALE

48966 None.
 48967

FUTURE DIRECTIONS

48968 None.
 48969

SEE ALSO

48971 *close()*, *dup()*, *exec*, *fcntl()*, *fstat()*, *ftruncate()*, *mmap()*, *msync()*, *posix_mem_offset()*,
 48972 *posix_typed_mem_get_info()*, *umask()*

48973 XBD <fcntl.h>, <sys/mman.h>

CHANGE HISTORY

48974 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.
 48975

Issue 7

48976 Austin Group Interpretation 1003.1-2001 #143 is applied.
 48977

pow()*System Interfaces*48978 **NAME**48979 `pow, powf, powl` — power function48980 **SYNOPSIS**48981 `#include <math.h>`48982 `double pow(double x, double y);`48983 `float powf(float x, float y);`48984 `long double powl(long double x, long double y);`48985 **DESCRIPTION**

48986 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 48987 conflict between the requirements described here and the ISO C standard is unintentional. This
 48988 volume of POSIX.1-2008 defers to the ISO C standard.

48989 These functions shall compute the value of x raised to the power y , x^y . If x is negative, the
 48990 application shall ensure that y is an integer value.

48991 An application wishing to check for error situations should set `errno` to zero and call
 48992 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or
 48993 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-
 48994 zero, an error has occurred.

48995 **RETURN VALUE**48996 Upon successful completion, these functions shall return the value of x raised to the power y .

48997 **MX** For finite values of $x < 0$, and finite non-integer values of y , a domain error shall occur and
 48998 either a NaN (if representable), or an implementation-defined value shall be returned.

48999 If the correct value would cause overflow, a range error shall occur and `pow()`, `powf()`, and
 49000 `powl()` shall return `±HUGE_VAL`, `±HUGE_VALF`, and `±HUGE_VALL`, respectively, with the
 49001 same sign as the correct value of the function.

49002 If the correct value would cause underflow, and is not representable, a range error may occur,
 49003 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

49004 **CX** For $y < 0$, if x is zero, a pole error may occur and `pow()`, `powf()`, and `powl()` shall return
 49005 **MX** `±HUGE_VAL`, `±HUGE_VALF`, and `±HUGE_VALL`, respectively. On systems that support the
 49006 IEC 60559 Floating-Point option, a pole error shall occur and `pow()`, `powf()`, and `powl()` shall
 49007 return `±HUGE_VAL`, `±HUGE_VALF`, and `±HUGE_VALL`, respectively if y is an odd integer, or
 49008 `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL`, respectively if y is not an odd integer.

49009 **MX** If x or y is a NaN, a NaN shall be returned (unless specified elsewhere in this description).49010 For any value of y (including NaN), if x is +1, 1.0 shall be returned.49011 For any value of x (including NaN), if y is ± 0 , 1.0 shall be returned.49012 For any odd integer value of $y > 0$, if x is ± 0 , ± 0 shall be returned.49013 For $y > 0$ and not an odd integer, if x is ± 0 , +0 shall be returned.49014 If x is -1, and y is $\pm \text{Inf}$, 1.0 shall be returned.49015 For $|x| < 1$, if y is -Inf, +Inf shall be returned.49016 For $|x| > 1$, if y is -Inf, +0 shall be returned.49017 For $|x| < 1$, if y is +Inf, +0 shall be returned.49018 For $|x| > 1$, if y is +Inf, +Inf shall be returned.49019 For y an odd integer < 0 , if x is -Inf, -0 shall be returned.

49020 For $y < 0$ and not an odd integer, if x is $-\text{Inf}$, $+0$ shall be returned.

49021 For y an odd integer > 0 , if x is $-\text{Inf}$, $-\text{Inf}$ shall be returned.

49022 For $y > 0$ and not an odd integer, if x is $-\text{Inf}$, $+\text{Inf}$ shall be returned.

49023 For $y < 0$, if x is $+\text{Inf}$, $+0$ shall be returned.

49024 For $y > 0$, if x is $+\text{Inf}$, $+\text{Inf}$ shall be returned.

49025 If the correct value would cause underflow, and is representable, a range error may occur and

49026 the correct value shall be returned.

ERRORS

49027 These functions shall fail if:

49028 Domain Error The value of x is negative and y is a finite non-integer.

49029 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,

49030 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*

49031 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception

49032 shall be raised.

49033

49034 MX Pole Error The value of x is zero and y is negative.

49035 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,

49036 then *errno* shall be set to [ERANGE]. If the integer expression

49037 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero

49038 floating-point exception shall be raised.

49039 Range Error The result overflows.

49040 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,

49041 then *errno* shall be set to [ERANGE]. If the integer expression

49042 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow

49043 floating-point exception shall be raised.

49044 These functions may fail if:

49045 Pole Error The value of x is zero and y is negative.

49046 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,

49047 then *errno* shall be set to [ERANGE]. If the integer expression

49048 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero

49049 floating-point exception shall be raised.

49050 Range Error The result underflows.

49051 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,

49052 then *errno* shall be set to [ERANGE]. If the integer expression

49053 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow

49054 floating-point exception shall be raised.

pow()*System Interfaces***EXAMPLES**

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO*exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*XBD Section 4.19 (on page 116), **<math.h>****CHANGE HISTORY**

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

The *powf()* and *powl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/42 is applied, correcting the third paragraph in the RETURN VALUE section.

Issue 7

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #51 (SD5-XSH-ERN-81) is applied.

*System Interfaces***pread()**49083 **NAME**49084

```
pread — read from a file
```

49085 **SYNOPSIS**49086

```
#include <unistd.h>
```

49087

```
ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
```

49088 **DESCRIPTION**49089 Refer to *read()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

printf()*System Interfaces*49090 **NAME**

49091 printf — print formatted output

49092 **SYNOPSIS**

49093 #include <stdio.h>

49094 int printf(const char *restrict *format*, ...);49095 **DESCRIPTION**49096 Refer to *fprintf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

pselect, select — synchronous I/O multiplexing

SYNOPSIS

```
#include <sys/select.h>

int pselect(int nfd, fd_set *restrict readfds,
            fd_set *restrict writefds, fd_set *restrict errorfds,
            const struct timespec *restrict timeout,
            const sigset_t *restrict sigmask);
int select(int nfd, fd_set *restrict readfds,
           fd_set *restrict writefds, fd_set *restrict errorfds,
           struct timeval *restrict timeout);
void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

DESCRIPTION

The *pselect()* function shall examine the file descriptor sets whose addresses are passed in the *readfds*, *writefds*, and *errorfds* parameters to see whether some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively.

The *select()* function shall be equivalent to the *pselect()* function, except as follows:

- For the *select()* function, the timeout period is given in seconds and microseconds in an argument of type **struct timeval**, whereas for the *pselect()* function the timeout period is given in seconds and nanoseconds in an argument of type **struct timespec**.
- The *select()* function has no *sigmask* argument; it shall behave as *pselect()* does when *sigmask* is a null pointer.
- Upon successful completion, the *select()* function may modify the object pointed to by the *timeout* argument.

OB XSR The *pselect()* and *select()* functions shall support regular files, terminal and pseudo-terminal devices, **STREAMS**-based files, FIFOs, pipes, and sockets. The behavior of *pselect()* and *select()* on file descriptors that refer to other types of file is unspecified.

The *nfd* argument specifies the range of descriptors to be tested. The first *nfd* descriptors shall be checked in each set; that is, the descriptors from zero through *nfd*–1 in the descriptor sets shall be examined.

If the *readfds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for being ready to read, and on output indicates which file descriptors are ready to read.

If the *writefds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for being ready to write, and on output indicates which file descriptors are ready to write.

If the *errorfds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for error conditions pending, and on output indicates which file descriptors have error conditions pending.

Upon successful completion, the *pselect()* or *select()* function shall modify the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively, and shall return the total number of ready descriptors in all the output sets. For each file descriptor less than *nfd*, the

pselect()*System Interfaces*

corresponding bit shall be set upon successful completion if it was set on input and the associated condition is true for that file descriptor.

If none of the selected descriptors are ready for the requested operation, the *pselect()* or *select()* function shall block until at least one of the requested operations becomes ready, until the *timeout* occurs, or until interrupted by a signal. The *timeout* parameter controls how long the *pselect()* or *select()* function shall take before timing out. If the *timeout* parameter is not a null pointer, it specifies a maximum interval to wait for the selection to complete. If the specified time interval expires without any requested operation becoming ready, the function shall return. If the *timeout* parameter is a null pointer, then the call to *pselect()* or *select()* shall block indefinitely until at least one descriptor meets the specified criteria. To effect a poll, the *timeout* parameter should not be a null pointer, and should point to a zero-valued **timespec** structure.

The use of a timeout does not affect any pending timers set up by *alarm()* or *setitimer()*.

Implementations may place limitations on the maximum timeout interval supported. All implementations shall support a maximum timeout interval of at least 31 days. If the *timeout* argument specifies a timeout interval greater than the implementation-defined maximum value, the maximum value shall be used as the actual timeout value. Implementations may also place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval shall be rounded up to the next supported value.

If *sigmask* is not a null pointer, then the *pselect()* function shall replace the signal mask of the caller by the set of signals pointed to by *sigmask* before examining the descriptors, and shall restore the signal mask of the calling thread before returning.

A descriptor shall be considered ready for reading when a call to an input function with **O_NONBLOCK** clear would not block, whether or not the function would transfer data successfully. (The function might return data, an end-of-file indication, or an error other than one indicating that it is blocked, and in each of these cases the descriptor shall be considered ready for reading.)

A descriptor shall be considered ready for writing when a call to an output function with **O_NONBLOCK** clear would not block, whether or not the function would transfer data successfully.

If a socket has a pending error, it shall be considered to have an exceptional condition pending. Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for use with a socket, it is protocol-specific except as noted below. For other file types it is implementation-defined. If the operation is meaningless for a particular file type, *pselect()* or *select()* shall indicate that the descriptor is ready for read or write operations, and shall indicate that the descriptor has no exceptional condition pending.

If a descriptor refers to a socket, the implied input function is the *recvmsg()* function with parameters requesting normal and ancillary data, such that the presence of either type shall cause the socket to be marked as readable. The presence of out-of-band data shall be checked if the socket option **SO_OOBINLINE** has been enabled, as out-of-band data is enqueued with normal data. If the socket is currently listening, then it shall be marked as readable if an incoming connection request has been received, and a call to the *accept()* function shall complete without blocking.

If a descriptor refers to a socket, the implied output function is the *sendmsg()* function supplying an amount of normal data equal to the current value of the **SO_SNDLOWAT** option for the socket. If a non-blocking call to the *connect()* function has been made for a socket, and the connection attempt has either succeeded or failed leaving a pending error, the socket shall be

marked as writable.

A socket shall be considered to have an exceptional condition pending if a receive operation with `O_NONBLOCK` clear for the open file description and with the `MSG_OOB` flag set would return out-of-band data without blocking. (It is protocol-specific whether the `MSG_OOB` flag would be used to read out-of-band data.) A socket shall also be considered to have an exceptional condition pending if an out-of-band data mark is present in the receive queue. Other circumstances under which a socket may be considered to have an exceptional condition pending are protocol-specific and implementation-defined.

If the *readfds*, *writefds*, and *errorfds* arguments are all null pointers and the *timeout* argument is not a null pointer, the *pselect()* or *select()* function shall block for the time specified, or until interrupted by a signal. If the *readfds*, *writefds*, and *errorfds* arguments are all null pointers and the *timeout* argument is a null pointer, the *pselect()* or *select()* function shall block until interrupted by a signal.

File descriptors associated with regular files shall always select true for ready to read, ready to write, and error conditions.

On failure, the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments shall not be modified. If the timeout interval expires without the specified condition being true for any of the specified file descriptors, the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments shall have all bits set to 0.

File descriptor masks of type **fd_set** can be initialized and tested with *FD_CLR()*, *FD_ISSET()*, *FD_SET()*, and *FD_ZERO()*. It is unspecified whether each of these is a macro or a function. If a macro definition is suppressed in order to access an actual function, or a program defines an external identifier with any of these names, the behavior is undefined.

FD_CLR(fd, fdsetp) shall remove the file descriptor *fd* from the set pointed to by *fdsetp*. If *fd* is not a member of this set, there shall be no effect on the set, nor will an error be returned.

FD_ISSET(fd, fdsetp) shall evaluate to non-zero if the file descriptor *fd* is a member of the set pointed to by *fdsetp*, and shall evaluate to zero otherwise.

FD_SET(fd, fdsetp) shall add the file descriptor *fd* to the set pointed to by *fdsetp*. If the file descriptor *fd* is already in this set, there shall be no effect on the set, nor will an error be returned.

FD_ZERO(fdsetp) shall initialize the descriptor set pointed to by *fdsetp* to the null set. No error is returned if the set is not empty at the time *FD_ZERO()* is invoked.

The behavior of these macros is undefined if the *fd* argument is less than 0 or greater than or equal to `FD_SETSIZE`, or if *fd* is not a valid file descriptor, or if any of the arguments are expressions with side-effects.

If a thread gets canceled during a *pselect()* call, the signal mask in effect when executing the registered cleanup functions is either the original signal mask or the signal mask installed as part of the *pselect()* call.

RETURN VALUE

Upon successful completion, the *pselect()* and *select()* functions shall return the total number of bits set in the bit masks. Otherwise, -1 shall be returned, and *errno* shall be set to indicate the error.

FD_CLR(), *FD_SET()*, and *FD_ZERO()* do not return a value. *FD_ISSET()* shall return a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and 0 otherwise.

pselect()*System Interfaces***ERRORS**

Under the following conditions, *pselect()* and *select()* shall fail and set *errno* to:

49237	[EBADF]	One or more of the file descriptor sets specified a file descriptor that is not a valid open file descriptor.
49239	[EINTR]	The function was interrupted before any of the selected events occurred and before the timeout interval expired.
49241 XSI		If SA_RESTART has been set for the interrupting signal, it is implementation-defined whether the function restarts or returns with [EINTR].
49243	[EINVAL]	An invalid timeout interval was specified.
49244	[EINVAL]	The <i>nfds</i> argument is less than 0 or greater than FD_SETSIZE.
49245 OB XSR	[EINVAL]	One of the specified file descriptors refers to a STREAM or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

In earlier versions of the Single UNIX Specification, the *select()* function was defined in the `<sys/time.h>` header. This is now changed to `<sys/select.h>`. The rationale for this change was as follows: the introduction of the *pselect()* function included the `<sys/select.h>` header and the `<sys/select.h>` header defines all the related definitions for the *pselect()* and *select()* functions. Backwards-compatibility to existing XSI implementations is handled by allowing `<sys/time.h>` to include `<sys/select.h>`.

Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers can install an additional cancellation handler which resets the signal mask to the expected value.

```

void cleanup(void *arg)
{
    sigset_t *ss = (sigset_t *) arg;
    pthread_sigmask(SIG_SETMASK, ss, NULL);
}

int call_pselect(int nfds, fd_set *readfds, fd_set *writefds,
    fd_set errorfds, const struct timespec *timeout,
    const sigset_t *sigmask)
{
    sigset_t oldmask;
    int result;
    pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
    pthread_cleanup_push(cleanup, &oldmask);
    result = pselect(nfds, readfds, writefds, errorfds, timeout, sigmask);
    pthread_cleanup_pop(0);
    return result;
}

```

FUTURE DIRECTIONS

None.

SEE ALSO

accept(), *alarm()*, *connect()*, *fcntl()*, *getitimer()*, *poll()*, *read()*, *recvmsg()*, *sendmsg()*, *write()*

XBD **<sys/select.h>**, **<sys/time.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

In the ERRORS section, the text has been changed to indicate that [EINVAL] is returned when *nfds* is less than 0 or greater than FD_SETSIZE. It previously stated less than 0, or greater than or equal to FD_SETSIZE.

Text about *timeout* is moved from the APPLICATION USAGE section to the DESCRIPTION.

Issue 6

The Open Group Corrigendum U026/6 is applied, changing the occurrences of *readfs* and *writefs* in the *select()* DESCRIPTION to be *readfds* and *writefds*.

Text referring to sockets is added to the DESCRIPTION.

The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- These functions are now mandatory.

The *pselect()* function is added for alignment with IEEE Std 1003.1g-2000 and additional detail related to sockets semantics is added to the DESCRIPTION.

The *select()* function now requires inclusion of **<sys/select.h>**.

The **restrict** keyword is added to the *select()* prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/70 is applied, updating the DESCRIPTION to reference the signal mask in terms of the calling thread rather than the process.

Issue 7

SD5-XSH-ERN-122 is applied, adding text to the DESCRIPTION for when a thread is canceled during a call to *pselect()*, and adding example code to the RATIONALE.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Functionality relating to the Threads option is moved to the Base.

psiginfo()

System Interfaces

49312 **NAME**

49313 psiginfo, psignal — print signal information to standard error

49314 **SYNOPSIS**

```

49315 CX      #include <signal.h>
49316          void psiginfo(const siginfo_t *pinfo, const char *message);
49317          void psignal(int signum, const char *message);

```

49318 **DESCRIPTION**

49319 The *psiginfo()* and *psignal()* functions shall print a message out on *stderr* associated with a signal number. If *message* is not null and is not the empty string, then the string pointed to by the *message* argument shall be printed first, followed by a <colon>, a <space> and the signal description string indicated by *signum*, or by the signal associated with *pinfo*. If the *message* argument is null or points to an empty string, then only the signal description shall be printed. For *psiginfo()*, the argument *pinfo* references a valid **siginfo_t** structure. For *psignal()*, if *signum* is not a valid signal number, the behavior is implementation-defined.

49326 The *psiginfo()* and *psignal()* functions shall not change the orientation of the standard error stream.

49328 The *psiginfo()* and *psignal()* functions shall mark for update the last data modification and last file status change timestamps of the file associated with the standard error stream at some time between their successful completion and *exit()*, *abort()*, or the completion of *fflush()* or *fclose()* on *stderr*.

49332 **RETURN VALUE**

49333 These functions shall not return a value.

49334 **ERRORS**

49335 No errors are defined.

49336 **EXAMPLES**

49337 None.

49338 **APPLICATION USAGE**

49339 None.

49340 **RATIONALE**

49341 System V historically has *psignal()* and *psiginfo()* in <**siginfo.h**>. However, the <**siginfo.h**> header is not specified in the Base Definitions volume of POSIX.1-2008, and the type **siginfo_t** is defined in <**signal.h**>.

49344 **FUTURE DIRECTIONS**

49345 None.

49346 **SEE ALSO**49347 *perror()*, *strsignal()*49348 XBD <**signal.h**>49349 **CHANGE HISTORY**

49350 First released in Issue 7.

System Interfaces

pthread_atfork()**NAME**

pthread_atfork — register fork handlers

SYNOPSIS

```
#include <pthread.h>
```

```
int pthread_atfork(void (*prepare)(void), void (*parent)(void),
void (*child)(void));
```

DESCRIPTION

The *pthread_atfork()* function shall declare fork handlers to be called before and after *fork()*, in the context of the thread that called *fork()*. The *prepare* fork handler shall be called before *fork()* processing commences. The *parent* fork handler shall be called after *fork()* processing completes in the parent process. The *child* fork handler shall be called after *fork()* processing completes in the child process. If no handling is desired at one or more of these three points, the corresponding fork handler address(es) may be set to NULL.

The order of calls to *pthread_atfork()* is significant. The *parent* and *child* fork handlers shall be called in the order in which they were established by calls to *pthread_atfork()*. The *prepare* fork handlers shall be called in the opposite order.

RETURN VALUE

Upon successful completion, *pthread_atfork()* shall return a value of zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_atfork()* function shall fail if:

[ENOMEM] Insufficient table space exists to record the fork handler addresses.

The *pthread_atfork()* function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

There are at least two serious problems with the semantics of *fork()* in a multi-threaded program. One problem has to do with state (for example, memory) covered by mutexes. Consider the case where one thread has a mutex locked and the state covered by that mutex is inconsistent while another thread calls *fork()*. In the child, the mutex is in the locked state (locked by a nonexistent thread and thus can never be unlocked). Having the child simply reinitialize the mutex is unsatisfactory since this approach does not resolve the question about how to correct or otherwise deal with the inconsistent state in the child.

It is suggested that programs that use *fork()* call an *exec* function very soon afterwards in the child process, thus resetting all states. In the meantime, only a short list of async-signal-safe library routines are promised to be available.

Unfortunately, this solution does not address the needs of multi-threaded libraries. Application programs may not be aware that a multi-threaded library is in use, and they feel free to call any number of library routines between the *fork()* and *exec* calls, just as they always have. Indeed, they may be extant single-threaded programs and cannot, therefore, be expected to obey new restrictions imposed by the threads library.

On the other hand, the multi-threaded library needs a way to protect its internal state during *fork()* in case it is re-entered later in the child process. The problem arises especially in multi-

pthread_atfork()*System Interfaces*

threaded I/O libraries, which are almost sure to be invoked between the *fork()* and *exec* calls to effect I/O redirection. The solution may require locking mutex variables during *fork()*, or it may entail simply resetting the state in the child after the *fork()* processing completes.

The *pthread_atfork()* function was intended to provide multi-threaded libraries with a means to protect themselves from innocent application programs that call *fork()*, and to provide multi-threaded application programs with a standard mechanism for protecting themselves from *fork()* calls in a library routine or the application itself.

The expected usage was that the prepare handler would acquire all mutex locks and the other two fork handlers would release them.

For example, an application could have supplied a prepare routine that acquires the necessary mutexes the library maintains and supplied child and parent routines that release those mutexes, thus ensuring that the child would have got a consistent snapshot of the state of the library (and that no mutexes would have been left stranded). This is good in theory, but in reality not practical. Each and every mutex and lock in the process must be located and locked. Every component of a program including third-party components must participate and they must agree who is responsible for which mutex or lock. This is especially problematic for mutexes and locks in dynamically allocated memory. All mutexes and locks internal to the implementation must be locked, too. This possibly delays the thread calling *fork()* for a long time or even indefinitely since uses of these synchronization objects may not be under control of the application. A final problem to mention here is the problem of locking streams. At least the streams under control of the system (like *stdin*, *stdout*, *stderr*) must be protected by locking the stream with *flockfile()*. But the application itself could have done that, possibly in the same thread calling *fork()*. In this case, the process will deadlock.

Alternatively, some libraries might have been able to supply just a *child* routine that reinitializes the mutexes in the library and all associated states to some known value (for example, what it was when the image was originally executed). This approach is not possible, though, because implementations are allowed to fail **_init()* and **_destroy()* calls for mutexes and locks if the mutex or lock is still locked. In this case, the *child* routine is not able to reinitialize the mutexes and locks.

When *fork()* is called, only the calling thread is duplicated in the child process. Synchronization variables remain in the same state in the child as they were in the parent at the time *fork()* was called. Thus, for example, mutex locks may be held by threads that no longer exist in the child process, and any associated states may be inconsistent. The intention was that the parent process could have avoided this by explicit code that acquires and releases locks critical to the child via *pthread_atfork()*. In addition, any critical threads would have needed to be recreated and reinitialized to the proper state in the child (also via *pthread_atfork()*).

A higher-level package may acquire locks on its own data structures before invoking lower-level packages. Under this scenario, the order specified for fork handler calls allows a simple rule of initialization for avoiding package deadlock: a package initializes all packages on which it depends before it calls the *pthread_atfork()* function for itself.

As explained, there is no suitable solution for functionality which requires non-atomic operations to be protected through mutexes and locks. This is why the POSIX.1 standard since the 1996 release requires that the child process after *fork()* in a multi-threaded process only calls *async-signal-safe* interfaces.

49440 **FUTURE DIRECTIONS**

49441 None.

49442 **SEE ALSO**49443 *atexit()*, *exec*, *fork()*49444 XBD *<pthread.h>*, *<sys/types.h>*49445 **CHANGE HISTORY**

49446 First released in Issue 5. Derived from the POSIX Threads Extension.

49447 IEEE PASC Interpretation 1003.1c #4 is applied.

49448 **Issue 6**49449 The *pthread_atfork()* function is marked as part of the Threads option.49450 The *<pthread.h>* header is added to the SYNOPSIS.49451 **Issue 7**49452 The *pthread_atfork()* function is moved from the Threads option to the Base.

49453 SD5-XSH-ERN-145 is applied, updating the RATIONALE.

pthread_attr_destroy()

System Interfaces

NAME

pthread_attr_destroy, pthread_attr_init — destroy and initialize the thread attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_init(pthread_attr_t *attr);
```

DESCRIPTION

The *pthread_attr_destroy()* function shall destroy a thread attributes object. An implementation may cause *pthread_attr_destroy()* to set *attr* to an implementation-defined invalid value. A destroyed *attr* attributes object can be reinitialized using *pthread_attr_init()*; the results of otherwise referencing the object after it has been destroyed are undefined.

The *pthread_attr_init()* function shall initialize a thread attributes object *attr* with the default value for all of the individual attributes used by a given implementation.

The resulting attributes object (possibly modified by setting individual attribute values) when used by *pthread_create()* defines the attributes of the thread created. A single attributes object can be used in multiple simultaneous calls to *pthread_create()*. Results are undefined if *pthread_attr_init()* is called specifying an already initialized *attr* attributes object.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_destroy()* does not refer to an initialized thread attributes object.

RETURN VALUE

Upon successful completion, *pthread_attr_destroy()* and *pthread_attr_init()* shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_init()* function shall fail if:

[ENOMEM] Insufficient memory exists to initialize the thread attributes object.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to support probable future standardization in these areas without requiring that the function itself be changed.

Attributes objects provide clean isolation of the configurable aspects of threads. For example, “stack size” is an important attribute of a thread, but it cannot be expressed portably. When porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects can help by allowing the changes to be isolated in a single place, rather than being spread across every instance of thread creation.

Attributes objects can be used to set up “classes” of threads with similar attributes; for example, “threads with large stacks and high priority” or “threads with minimal stacks”. These classes can be defined in a single place and then referenced wherever threads need to be created. Changes to “class” decisions become straightforward, and detailed analysis of each *pthread_create()* call is not required.

The attributes objects are defined as opaque types as an aid to extensibility. If these objects had been specified as structures, adding new attributes would force recompilation of all multi-threaded programs when the attributes objects are extended; this might not be possible if different program components were supplied by different vendors.

Additionally, opaque attributes objects present opportunities for improving performance. Argument validity can be checked once when attributes are set, rather than each time a thread is created. Implementations often need to cache kernel objects that are expensive to create. Opaque attributes objects provide an efficient mechanism to detect when cached objects become invalid due to attribute changes.

Since assignment is not necessarily defined on a given opaque type, implementation-defined default values cannot be defined in a portable way. The solution to this problem is to allow attributes objects to be initialized dynamically by attributes object initialization functions, so that default values can be supplied automatically by the implementation.

The following proposal was provided as a suggested alternative to the supplied attributes:

1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to the initialization routines (*pthread_create()*, *pthread_mutex_init()*, *pthread_cond_init()*). The parameter containing the flags should be an opaque type for extensibility. If no flags are set in the parameter, then the objects are created with default characteristics. An implementation may specify implementation-defined flag values and associated behavior.
2. If further specialization of mutexes and condition variables is necessary, implementations may specify additional procedures that operate on the **pthread_mutex_t** and **pthread_cond_t** objects (instead of on attributes objects).

The difficulties with this solution are:

1. A bitmask is not opaque if bits have to be set into bitvector attributes objects using explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**, application programmers need to know the location of each bit. If bits are set or read by encapsulation (that is, get and set functions), then the bitmask is merely an implementation of attributes objects as currently defined and should not be exposed to the programmer.
2. Many attributes are not Boolean or very small integral values. For example, scheduling policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this, the bitmask can only reasonably control whether particular attributes are set or not, and it cannot serve as the repository of the value itself. The value needs to be specified as a function parameter (which is non-extensible), or by setting a structure field (which is non-opaque), or by get and set functions (making the bitmask a redundant addition to the attributes objects).

Stack size is defined as an optional attribute because the very notion of a stack is inherently machine-dependent. Some implementations may not be able to change the size of the stack, for example, and others may not need to because stack pages may be discontinuous and can be allocated and released on demand.

The attribute mechanism has been designed in large measure for extensibility. Future extensions to the attribute mechanism or to any attributes object defined in this volume of POSIX.1-2008 has to be done with care so as not to affect binary-compatibility.

Attributes objects, even if allocated by means of dynamic allocation functions such as *malloc()*,

pthread_attr_destroy()*System Interfaces*

may have their size fixed at compile time. This means, for example, a *pthread_create()* in an implementation with extensions to **pthread_attr_t** cannot look beyond the area that the binary application assumes is valid. This suggests that implementations should maintain a size field in the attributes object, as well as possibly version information, if extensions in different directions (possibly by different vendors) are to be accommodated.

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_destroy()* does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_init()* refers to an already initialized thread attributes object, it is recommended that the function should fail and report an [EBUSY] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_getstacksize(), *pthread_attr_getdetachstate()*, *pthread_create()*

XBD <pthread.h>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_attr_destroy()* and *pthread_attr_init()* functions are marked as part of the Threads option.

IEEE PASC Interpretation 1003.1 #107 is applied, noting that the effect of initializing an already initialized thread attributes object is undefined.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/71 is applied, updating the ERRORS section to add the optional [EINVAL] error for the *pthread_attr_destroy()* function, and the optional [EBUSY] error for the *pthread_attr_init()* function.

Issue 7

The *pthread_attr_destroy()* and *pthread_attr_init()* functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for an already initialized thread attributes object is removed; this condition results in undefined behavior.

NAME

pthread_attr_getdetachstate, pthread_attr_setdetachstate — get and set the detachstate attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_getdetachstate(const pthread_attr_t *attr,
                               int *detachstate);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

DESCRIPTION

The *detachstate* attribute controls whether the thread is created in a detached state. If the thread is created detached, then use of the ID of the newly created thread by the *pthread_detach()* or *pthread_join()* function is an error.

The *pthread_attr_getdetachstate()* and *pthread_attr_setdetachstate()* functions, respectively, shall get and set the *detachstate* attribute in the *attr* object.

For *pthread_attr_getdetachstate()*, *detachstate* shall be set to either PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

For *pthread_attr_setdetachstate()*, the application shall set *detachstate* to either PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

A value of PTHREAD_CREATE_DETACHED shall cause all threads created with *attr* to be in the detached state, whereas using a value of PTHREAD_CREATE_JOINABLE shall cause all threads created with *attr* to be in the joinable state. The default value of the *detachstate* attribute shall be PTHREAD_CREATE_JOINABLE.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getdetachstate()* or *pthread_attr_setdetachstate()* does not refer to an initialized thread attributes object.

RETURN VALUE

Upon successful completion, *pthread_attr_getdetachstate()* and *pthread_attr_setdetachstate()* shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

The *pthread_attr_getdetachstate()* function stores the value of the *detachstate* attribute in *detachstate* if successful.

ERRORS

The *pthread_attr_setdetachstate()* function shall fail if:

[EINVAL] The value of *detachstate* was not valid

These functions shall not return an error code of [EINTR].

EXAMPLES**Retrieving the detachstate Attribute**

This example shows how to obtain the *detachstate* attribute of a thread attribute object.

```
#include <pthread.h>

pthread_attr_t thread_attr;
int detachstate;
int rc;

/* code initializing thread_attr */
...
```

pthread_attr_getdetachstate()

System Interfaces

```

49619     rc = pthread_attr_getdetachstate (&thread_attr, &detachstate);
49620     if (rc!=0) {
49621         /* handle error */
49622         ...
49623     }
49624     else {
49625         /* legal values for detachstate are:
49626          * PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE
49627          */
49628         ...
49629     }

```

APPLICATION USAGE

49631 None.

RATIONALE

49633 If an implementation detects that the value specified by the *attr* argument to
 49634 *pthread_attr_getdetachstate()* or *pthread_attr_setdetachstate()* does not refer to an initialized thread
 49635 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

49637 None.

SEE ALSO

49639 *pthread_attr_destroy()*, *pthread_attr_getstacksize()*, *pthread_create()*

49640 XBD <pthread.h>

CHANGE HISTORY

49642 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

49644 The *pthread_attr_setdetachstate()* and *pthread_attr_getdetachstate()* functions are marked as part of
 49645 the Threads option.

49646 The normative text is updated to avoid use of the term “must” for application requirements.

49647 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/72 is applied, adding the example to the
 49648 EXAMPLES section.

49649 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/73 is applied, updating the ERRORS
 49650 section to include the optional [EINVAL] error.

Issue 7

49652 The *pthread_attr_setdetachstate()* and *pthread_attr_getdetachstate()* functions are moved from the
 49653 Threads option to the Base.

49654 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition
 49655 results in undefined behavior.

System Interfaces

pthread_attr_getguardsize()**NAME**

pthread_attr_getguardsize, pthread_attr_setguardsize — get and set the thread guardsize attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
                             size_t *restrict guardsize);
int pthread_attr_setguardsize(pthread_attr_t *attr,
                             size_t guardsize);
```

DESCRIPTION

The *pthread_attr_getguardsize()* function shall get the *guardsize* attribute in the *attr* object. This attribute shall be returned in the *guardsize* parameter.

The *pthread_attr_setguardsize()* function shall set the *guardsize* attribute in the *attr* object. The new value of this attribute shall be obtained from the *guardsize* parameter. If *guardsize* is zero, a guard area shall not be provided for threads created with *attr*. If *guardsize* is greater than zero, a guard area of at least size *guardsize* bytes shall be provided for each thread created with *attr*.

The *guardsize* attribute controls the size of the guard area for the created thread's stack. The *guardsize* attribute provides protection against overflow of the stack pointer. If a thread's stack is created with guard protection, the implementation allocates extra memory at the overflow end of the stack as a buffer against stack overflow of the stack pointer. If an application overflows into this buffer an error shall result (possibly in a SIGSEGV signal being delivered to the thread).

A conforming implementation may round up the value contained in *guardsize* to a multiple of the configurable system variable {PAGESIZE} (see <sys/mman.h>). If an implementation rounds up the value of *guardsize* to a multiple of {PAGESIZE}, a call to *pthread_attr_getguardsize()* specifying *attr* shall store in the *guardsize* parameter the guard size specified by the previous *pthread_attr_setguardsize()* function call.

The default value of the *guardsize* attribute is implementation-defined.

If the *stackaddr* attribute has been set (that is, the caller is allocating and managing its own thread stacks), the *guardsize* attribute shall be ignored and no protection shall be provided by the implementation. It is the responsibility of the application to manage stack overflow along with stack allocation and management in this case.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getguardsize()* or *pthread_attr_setguardsize()* does not refer to an initialized thread attributes object.

RETURN VALUE

If successful, the *pthread_attr_getguardsize()* and *pthread_attr_setguardsize()* functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall fail if:

[EINVAL] The parameter *guardsize* is invalid.

These functions shall not return an error code of [EINTR].

pthread_attr_getguardsize()

System Interfaces

49697 **EXAMPLES**49698 **Retrieving the guardsize Attribute**49699 This example shows how to obtain the *guardsize* attribute of a thread attribute object.

```

49700 #include <pthread.h>
49701 pthread_attr_t thread_attr;
49702 size_t guardsize;
49703 int rc;
49704 /* code initializing thread_attr */
49705 ...
49706 rc = pthread_attr_getguardsize (&thread_attr, &guardsize);
49707 if (rc != 0) {
49708     /* handle error */
49709     ...
49710 }
49711 else {
49712     if (guardsize > 0) {
49713         /* a guard area of at least guardsize bytes is provided */
49714         ...
49715     }
49716     else {
49717         /* no guard area provided */
49718         ...
49719     }
49720 }

```

49721 **APPLICATION USAGE**

49722 None.

49723 **RATIONALE**49724 The *guardsize* attribute is provided to the application for two reasons:

- 49725 1. Overflow protection can potentially result in wasted system resources. An application
49726 that creates a large number of threads, and which knows its threads never overflow their
49727 stack, can save system resources by turning off guard areas.
- 49728 2. When threads allocate large data structures on the stack, large guard areas may be needed
49729 to detect stack overflow.

49730 The default size of the guard area is left implementation-defined since on systems supporting
49731 very large page sizes, the overhead might be substantial if at least one guard page is required by
49732 default.

49733 If an implementation detects that the value specified by the *attr* argument to
49734 *pthread_attr_getguardsize()* or *pthread_attr_setguardsize()* does not refer to an initialized thread
49735 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

49736 **FUTURE DIRECTIONS**

49737 None.

SEE ALSOXBD [<pthread.h>](#), [<sys/mman.h>](#)**CHANGE HISTORY**

First released in Issue 5.

Issue 6

In the ERRORS section, a third [EINVAL] error condition is removed as it is covered by the second error condition.

The **restrict** keyword is added to the *pthread_attr_getguardsize()* prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/74 is applied, updating the ERRORS section to remove the [EINVAL] error ("The attribute *attr* is invalid."), and replacing it with the optional [EINVAL] error.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/76 is applied, adding the example to the EXAMPLES section.

Issue 7

SD5-XSH-ERN-111 is applied, removing the reference to the *stack* attribute in the DESCRIPTION.

SD5-XSH-ERN-175 is applied, updating the DESCRIPTION to note that the default size of the guard area is implementation-defined.

The *pthread_attr_getguardsize()* and *pthread_attr_setguardsize()* functions are moved from the XSI option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getinheritsched()

System Interfaces

49760 **NAME**

49761 pthread_attr_getinheritsched, pthread_attr_setinheritsched — get and set the inheritsched
 49762 attribute (**REALTIME THREADS**)

49763 **SYNOPSIS**

```
49764 TPS #include <pthread.h>
49765
49765 int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
49766 int *restrict inheritsched);
49767
49767 int pthread_attr_setinheritsched(pthread_attr_t *attr,
49768 int inheritsched);
```

49769 **DESCRIPTION**

49770 The *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions, respectively, shall
 49771 get and set the *inheritsched* attribute in the *attr* argument.

49772 When the attributes objects are used by *pthread_create()*, the *inheritsched* attribute determines
 49773 how the other scheduling attributes of the created thread shall be set.

49774 The supported values of *inheritsched* shall be:

49775 **PTHREAD_INHERIT_SCHED**

49776 Specifies that the thread scheduling attributes shall be inherited from the creating thread,
 49777 and the scheduling attributes in this *attr* argument shall be ignored.

49778 **PTHREAD_EXPLICIT_SCHED**

49779 Specifies that the thread scheduling attributes shall be set to the corresponding values from
 49780 this attributes object.

49781 The symbols **PTHREAD_INHERIT_SCHED** and **PTHREAD_EXPLICIT_SCHED** are defined in
 49782 the **<pthread.h>** header.

49783 The following thread scheduling attributes defined by POSIX.1-2008 are affected by the
 49784 *inheritsched* attribute: scheduling policy (*schedpolicy*), scheduling parameters (*schedparam*), and
 49785 scheduling contention scope (*contentionscope*).

49786 The behavior is undefined if the value specified by the *attr* argument to
 49787 *pthread_attr_getinheritsched()* or *pthread_attr_setinheritsched()* does not refer to an initialized
 49788 thread attributes object.

49789 **RETURN VALUE**

49790 If successful, the *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions shall
 49791 return zero; otherwise, an error number shall be returned to indicate the error.

49792 **ERRORS**

49793 The *pthread_attr_setinheritsched()* function may fail if:

49794 [EINVAL] The value of *inheritsched* is not valid.

49795 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

49796 These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

After these attributes have been set, a thread can be created with the specified attributes using *pthread_create()*. Using these routines does not affect the current running thread.

See [Section 2.9.4](#) (on page 509) for further details on thread scheduling attributes and their default settings.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getinheritsched()* or *pthread_attr_setinheritsched()* does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy(), *pthread_attr_getscope()*, *pthread_attr_getschedpolicy()*,
pthread_attr_getschedparam(), *pthread_create()*

XBD [<pthread.h>](#), [<sched.h>](#)

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6

The *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions are marked as part of the Threads and Thread Execution Scheduling options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The **restrict** keyword is added to the *pthread_attr_getinheritsched()* prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/75 is applied, clarifying the values of *inheritsched* in the DESCRIPTION and adding two optional [EINVAL] errors to the ERRORS section for checking when *attr* refers to an uninitialized thread attribute object.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/77 is applied, adding a reference to [Section 2.9.4](#) (on page 509) in the APPLICATION USAGE section.

Issue 7

The *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions are moved from the Threads option.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getschedparam()

System Interfaces

49835 **NAME**

49836 pthread_attr_getschedparam, pthread_attr_setschedparam — get and set the schedparam
 49837 attribute

49838 **SYNOPSIS**

```
49839 #include <pthread.h>

49840 int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
49841                               struct sched_param *restrict param);
49842 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
49843                               const struct sched_param *restrict param);
```

49844 **DESCRIPTION**

49845 The *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions, respectively, shall
 49846 get and set the scheduling parameter attributes in the *attr* argument. The contents of the *param*
 49847 structure are defined in the **<sched.h>** header. For the SCHED_FIFO and SCHED_RR policies,
 49848 the only required member of *param* is *sched_priority*.

49849 TSP For the SCHED_SPORADIC policy, the required members of the *param* structure are
 49850 *sched_priority*, *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and
 49851 *sched_ss_max_repl*. The specified *sched_ss_repl_period* must be greater than or equal to the
 49852 specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.
 49853 The value of *sched_ss_max_repl* shall be within the inclusive range [1, {SS_REPL_MAX}] for the
 49854 function to succeed; if not, the function shall fail. It is unspecified whether the
 49855 *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are
 49856 rounded to align with the resolution of the clock being used.

49857 The behavior is undefined if the value specified by the *attr* argument to
 49858 *pthread_attr_getschedparam()* or *pthread_attr_setschedparam()* does not refer to an initialized thread
 49859 attributes object.

49860 **RETURN VALUE**

49861 If successful, the *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions shall
 49862 return zero; otherwise, an error number shall be returned to indicate the error.

49863 **ERRORS**

49864 The *pthread_attr_setschedparam()* function may fail if:

49865 [EINVAL] The value of *param* is not valid.

49866 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

49867 These functions shall not return an error code of [EINTR].

49868 **EXAMPLES**

49869 None.

49870 **APPLICATION USAGE**

49871 After these attributes have been set, a thread can be created with the specified attributes using
 49872 *pthread_create()*. Using these routines does not affect the current running thread.

49873 **RATIONALE**

49874 If an implementation detects that the value specified by the *attr* argument to
 49875 *pthread_attr_getschedparam()* or *pthread_attr_setschedparam()* does not refer to an initialized thread
 49876 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy(), *pthread_attr_getscope()*, *pthread_attr_getinheritsched()*,
pthread_attr_getschedpolicy(), *pthread_create()*

XBD **<pthread.h>**, **<sched.h>****CHANGE HISTORY**

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions are marked as part of the Threads option.

The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/78 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

Issue 7

The *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions are moved from the Threads option to the Base.

Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getschedpolicy()

System Interfaces

49901 **NAME**

49902 pthread_attr_getschedpolicy, pthread_attr_setschedpolicy — get and set the schedpolicy
 49903 attribute (**REALTIME THREADS**)

49904 **SYNOPSIS**

```
49905 TPS #include <pthread.h>
49906
49907 int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
49908                               int *restrict policy);
49908 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

49909 **DESCRIPTION**

49910 The *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions, respectively, shall
 49911 get and set the *schedpolicy* attribute in the *attr* argument.

49912 The supported values of *policy* shall include SCHED_FIFO, SCHED_RR, and SCHED_OTHER,
 49913 which are defined in the **<sched.h>** header. When threads executing with the scheduling policy
 49914 TSP SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC are waiting on a mutex, they shall acquire
 49915 the mutex in priority order when the mutex is unlocked.

49916 The behavior is undefined if the value specified by the *attr* argument to
 49917 *pthread_attr_getschedpolicy()* or *pthread_attr_setschedpolicy()* does not refer to an initialized thread
 49918 attributes object.

49919 **RETURN VALUE**

49920 If successful, the *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions shall
 49921 return zero; otherwise, an error number shall be returned to indicate the error.

49922 **ERRORS**

49923 The *pthread_attr_setschedpolicy()* function may fail if:

49924 [EINVAL] The value of *policy* is not valid.

49925 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

49926 These functions shall not return an error code of [EINTR].

49927 **EXAMPLES**

49928 None.

49929 **APPLICATION USAGE**

49930 After these attributes have been set, a thread can be created with the specified attributes using
 49931 *pthread_create()*. Using these routines does not affect the current running thread.

49932 See [Section 2.9.4](#) (on page 509) for further details on thread scheduling attributes and their
 49933 default settings.

49934 **RATIONALE**

49935 If an implementation detects that the value specified by the *attr* argument to
 49936 *pthread_attr_getschedpolicy()* or *pthread_attr_setschedpolicy()* does not refer to an initialized thread
 49937 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

49938 **FUTURE DIRECTIONS**

49939 None.

49940 **SEE ALSO**

49941 *pthread_attr_destroy()*, *pthread_attr_getscope()*, *pthread_attr_getinheritsched()*,
 49942 *pthread_attr_getschedparam()*, *pthread_create()*

49943 XBD **<pthread.h>**, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6

The *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions are marked as part of the Threads and Thread Execution Scheduling options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_attr_getschedpolicy()* prototype for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/79 is applied, adding a reference to [Section 2.9.4](#) (on page 509) in the APPLICATION USAGE section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/80 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

Issue 7

The *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions are moved from the Threads option.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getscope()

System Interfaces

49965 **NAME**

49966 pthread_attr_getscope, pthread_attr_setscope — get and set the contentionscope attribute
 49967 (REALTIME THREADS)

49968 **SYNOPSIS**

```
49969 TPS #include <pthread.h>
49970
49971 int pthread_attr_getscope(const pthread_attr_t *restrict attr,
49972                          int *restrict contentionscope);
49972 int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

49973 **DESCRIPTION**

49974 The *pthread_attr_getscope()* and *pthread_attr_setscope()* functions, respectively, shall get and set
 49975 the *contentionscope* attribute in the *attr* object.

49976 The *contentionscope* attribute may have the values PTHREAD_SCOPE_SYSTEM, signifying
 49977 system scheduling contention scope, or PTHREAD_SCOPE_PROCESS, signifying process
 49978 scheduling contention scope. The symbols PTHREAD_SCOPE_SYSTEM and
 49979 PTHREAD_SCOPE_PROCESS are defined in the **<pthread.h>** header.

49980 The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getscope()* or
 49981 *pthread_attr_setscope()* does not refer to an initialized thread attributes object.

49982 **RETURN VALUE**

49983 If successful, the *pthread_attr_getscope()* and *pthread_attr_setscope()* functions shall return zero;
 49984 otherwise, an error number shall be returned to indicate the error.

49985 **ERRORS**

49986 The *pthread_attr_setscope()* function may fail if:

- 49987 [EINVAL] The value of *contentionscope* is not valid.
- 49988 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.
- 49989 These functions shall not return an error code of [EINTR].

49990 **EXAMPLES**

49991 None.

49992 **APPLICATION USAGE**

49993 After these attributes have been set, a thread can be created with the specified attributes using
 49994 *pthread_create()*. Using these routines does not affect the current running thread.

49995 See [Section 2.9.4](#) (on page 509) for further details on thread scheduling attributes and their
 49996 default settings.

49997 **RATIONALE**

49998 If an implementation detects that the value specified by the *attr* argument to
 49999 *pthread_attr_getscope()* or *pthread_attr_setscope()* does not refer to an initialized thread attributes
 50000 object, it is recommended that the function should fail and report an [EINVAL] error.

50001 **FUTURE DIRECTIONS**

50002 None.

50003 **SEE ALSO**

50004 *pthread_attr_destroy()*, *pthread_attr_getinheritsched()*, *pthread_attr_getschedpolicy()*,
 50005 *pthread_attr_getschedparam()*, *pthread_create()*

50006 XBD **<pthread.h>**, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6

The *pthread_attr_getscope()* and *pthread_attr_setscope()* functions are marked as part of the Threads and Thread Execution Scheduling options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The **restrict** keyword is added to the *pthread_attr_getscope()* prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/81 is applied, adding a reference to [Section 2.9.4](#) (on page 509) in the APPLICATION USAGE section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/82 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

Issue 7

The *pthread_attr_getscope()* and *pthread_attr_setscope()* functions are moved from the Threads option.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getstack()*System Interfaces*50027 **NAME**

50028 pthread_attr_getstack, pthread_attr_setstack — get and set stack attributes

50029 **SYNOPSIS**

```

50030 TSA TSS #include <pthread.h>
50031
50031 int pthread_attr_getstack(const pthread_attr_t *restrict attr,
50032 void **restrict stackaddr, size_t *restrict stacksize);
50033
50033 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
50034 size_t stacksize);

```

50035 **DESCRIPTION**

50036 The *pthread_attr_getstack()* and *pthread_attr_setstack()* functions, respectively, shall get and set the
 50037 thread creation stack attributes *stackaddr* and *stacksize* in the *attr* object.

50038 The stack attributes specify the area of storage to be used for the created thread's stack. The base
 50039 (lowest addressable byte) of the storage shall be *stackaddr*, and the size of the storage shall be
 50040 *stacksize* bytes. The *stacksize* shall be at least {PTHREAD_STACK_MIN}. The
 50041 *pthread_attr_setstack()* function may fail with [EINVAL] if *stackaddr* does not meet
 50042 implementation-defined alignment requirements. All pages within the stack described by
 50043 *stackaddr* and *stacksize* shall be both readable and writable by the thread.

50044 If the *pthread_attr_getstack()* function is called before the *stackaddr* attribute has been set, the
 50045 behavior is unspecified.

50046 The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getstack()* or
 50047 *pthread_attr_setstack()* does not refer to an initialized thread attributes object.

50048 **RETURN VALUE**

50049 Upon successful completion, these functions shall return a value of 0; otherwise, an error
 50050 number shall be returned to indicate the error.

50051 The *pthread_attr_getstack()* function shall store the stack attribute values in *stackaddr* and *stacksize*
 50052 if successful.

50053 **ERRORS**

50054 The *pthread_attr_setstack()* function shall fail if:

50055 [EINVAL] The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds an
 50056 implementation-defined limit.

50057 The *pthread_attr_setstack()* function may fail if:

50058 [EINVAL] The value of *stackaddr* does not have proper alignment to be used as a stack, or
 50059 ((**char** *)*stackaddr* + *stacksize*) lacks proper alignment.

50060 [EACCES] The stack page(s) described by *stackaddr* and *stacksize* are not both readable
 50061 and writable by the thread.

50062 These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

These functions are appropriate for use by applications in an environment where the stack for a thread must be placed in some particular region of memory.

While it might seem that an application could detect stack overflow by providing a protected page outside the specified stack region, this cannot be done portably. Implementations are free to place the thread's initial stack pointer anywhere within the specified region to accommodate the machine's stack pointer behavior and allocation requirements. Furthermore, on some architectures, such as the IA-64, "overflow" might mean that two separate stack pointers allocated within the region will overlap somewhere in the middle of the region.

After a successful call to *pthread_attr_setstack()*, the storage area specified by the *stackaddr* parameter is under the control of the implementation, as described in [Section 2.9.8](#) (on page 516).

The specification of the *stackaddr* attribute presents several ambiguities that make portable use of these functions impossible. For example, the standard allows implementations to impose arbitrary alignment requirements on *stackaddr*. Applications cannot assume that a buffer obtained from *malloc()* is suitably aligned. Note that although the *stacksize* value passed to *pthread_attr_setstack()* must satisfy alignment requirements, the same is not true for *pthread_attr_setstacksize()* where the implementation must increase the specified size if necessary to achieve the proper alignment.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getstack()* or *pthread_attr_setstack()* does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy(), *pthread_attr_getdetachstate()*, *pthread_attr_getstacksize()*, *pthread_create()*

XBD [<limits.h>](#), [<pthread.h>](#)

CHANGE HISTORY

First released in Issue 6. Developed as part of the XSI option and brought into the BASE by IEEE PASC Interpretation 1003.1 #101.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/83 is applied, updating the APPLICATION USAGE section to refer to [Section 2.9.8](#) (on page 516).

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC/D6/84 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

Issue 7

SD5-XSH-ERN-66 is applied, correcting the use of *attr* in the [EINVAL] error condition.

Austin Group Interpretation 1003.1-2001 #057 is applied, clarifying the behavior if the function is called before the *stackaddr* attribute is set.

SD5-XSH-ERN-157 is applied, updating the APPLICATION USAGE section.

The description of the *stackaddr* attribute is updated in the DESCRIPTION and APPLICATION USAGE sections.

pthread_attr_getstack()*System Interfaces*50107
50108

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

50109 NAME

50110 pthread_attr_getstacksize, pthread_attr_setstacksize — get and set the stacksize attribute

50111 SYNOPSIS

```
50112 TSS      #include <pthread.h>
50113
50113      int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
50114                                   size_t *restrict stacksize);
50115      int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

50116 DESCRIPTION

50117 The *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* functions, respectively, shall get and
 50118 set the thread creation *stacksize* attribute in the *attr* object.

50119 The *stacksize* attribute shall define the minimum stack size (in bytes) allocated for the created
 50120 threads stack.

50121 The behavior is undefined if the value specified by the *attr* argument to
 50122 *pthread_attr_getstacksize()* or *pthread_attr_setstacksize()* does not refer to an initialized thread
 50123 attributes object.

50124 RETURN VALUE

50125 Upon successful completion, *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* shall
 50126 return a value of 0; otherwise, an error number shall be returned to indicate the error.

50127 The *pthread_attr_getstacksize()* function stores the *stacksize* attribute value in *stacksize* if
 50128 successful.

50129 ERRORS

50130 The *pthread_attr_setstacksize()* function shall fail if:

50131 [EINVAL] The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds a
 50132 system-imposed limit.

50133 These functions shall not return an error code of [EINTR].

50134 EXAMPLES

50135 None.

50136 APPLICATION USAGE

50137 None.

50138 RATIONALE

50139 If an implementation detects that the value specified by the *attr* argument to
 50140 *pthread_attr_getstacksize()* or *pthread_attr_setstacksize()* does not refer to an initialized thread
 50141 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

50142 FUTURE DIRECTIONS

50143 None.

50144 SEE ALSO

50145 *pthread_attr_destroy()*, *pthread_attr_getdetachstate()*, *pthread_create()*

50146 XBD *<limits.h>*, *<pthread.h>*

50147 CHANGE HISTORY

50148 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

pthread_attr_getstacksize()

System Interfaces

Issue 6

The *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* functions are marked as part of the Threads and Thread Stack Size Attribute options.

The **restrict** keyword is added to the *pthread_attr_getstacksize()* prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/43 is applied, correcting the margin code in the SYNOPSIS from TSA to TSS and updating the CHANGE HISTORY from “Thread Stack Address Attribute” option to “Thread Stack Size Attribute” option.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/87 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

Issue 7

The *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* functions are moved from the Threads option.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

*System Interfaces***pthread_attr_init()**50165 **NAME**

50166 pthread_attr_init — initialize the thread attributes object

50167 **SYNOPSIS**

50168 #include <pthread.h>

50169 int pthread_attr_init(pthread_attr_t *attr);

50170 **DESCRIPTION**50171 Refer to *pthread_attr_destroy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_attr_setdetachstate()*System Interfaces*50172 **NAME**

50173 pthread_attr_setdetachstate — set the detachstate attribute

50174 **SYNOPSIS**

50175 #include <pthread.h>

50176 int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);

50177 **DESCRIPTION**50178 Refer to *pthread_attr_getdetachstate()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread_attr_setguardsize()**50179 **NAME**

50180 pthread_attr_setguardsize — set the thread guardsize attribute

50181 **SYNOPSIS**

50182 #include <pthread.h>

50183 int pthread_attr_setguardsize(pthread_attr_t *attr,
50184 size_t guardsize);50185 **DESCRIPTION**50186 Refer to *pthread_attr_getguardsize()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_attr_setinheritsched()*System Interfaces*50187 **NAME**50188 pthread_attr_setinheritsched — set the inheritsched attribute (**REALTIME THREADS**)50189 **SYNOPSIS**

```
50190 TPS #include <pthread.h>  
50191 int pthread_attr_setinheritsched(pthread_attr_t *attr,  
50192 int inheritsched);
```

50193 **DESCRIPTION**50194 Refer to *pthread_attr_getinheritsched()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread_attr_setschedparam()**50195 **NAME**

50196 pthread_attr_setschedparam — set the schedparam attribute

50197 **SYNOPSIS**

50198 #include <pthread.h>

50199 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
50200 const struct sched_param *restrict param);50201 **DESCRIPTION**50202 Refer to *pthread_attr_getschedparam()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_attr_setschedpolicy()*System Interfaces*50203 **NAME**50204 pthread_attr_setschedpolicy — set the schedpolicy attribute (**REALTIME THREADS**)50205 **SYNOPSIS**

```
50206 TPS #include <pthread.h>  
50207 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

50208 **DESCRIPTION**50209 Refer to *pthread_attr_getschedpolicy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread_attr_setscope()**50210 **NAME**50211 pthread_attr_setscope — set the contentionscope attribute (**REALTIME THREADS**)50212 **SYNOPSIS**

50213 TPS #include <pthread.h>

50214 int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);

50215 **DESCRIPTION**50216 Refer to *pthread_attr_getscope()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_attr_setstack()*System Interfaces*50217 **NAME**

50218 pthread_attr_setstack — set the stack attribute

50219 **SYNOPSIS**

```
50220 TSA TSS #include <pthread.h>  
50221 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,  
50222 size_t stacksize);
```

50223 **DESCRIPTION**50224 Refer to *pthread_attr_getstack()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread_attr_setstacksize()**50225 **NAME**

50226 pthread_attr_setstacksize — set the stacksize attribute

50227 **SYNOPSIS**

50228 TSS #include <pthread.h>

50229 int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);

50230 **DESCRIPTION**50231 Refer to *pthread_attr_getstacksize()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_barrier_destroy()

System Interfaces

50232 **NAME**

50233 pthread_barrier_destroy, pthread_barrier_init — destroy and initialize a barrier object

50234 **SYNOPSIS**

```
50235 #include <pthread.h>
50236
50236 int pthread_barrier_destroy(pthread_barrier_t *barrier);
50237 int pthread_barrier_init(pthread_barrier_t *restrict barrier,
50238     const pthread_barrierattr_t *restrict attr, unsigned count);
```

50239 **DESCRIPTION**

50240 The *pthread_barrier_destroy()* function shall destroy the barrier referenced by *barrier* and release
 50241 any resources used by the barrier. The effect of subsequent use of the barrier is undefined until
 50242 the barrier is reinitialized by another call to *pthread_barrier_init()*. An implementation may use
 50243 this function to set *barrier* to an invalid value. The results are undefined if
 50244 *pthread_barrier_destroy()* is called when any thread is blocked on the barrier, or if this function is
 50245 called with an uninitialized barrier.

50246 The *pthread_barrier_init()* function shall allocate any resources required to use the barrier
 50247 referenced by *barrier* and shall initialize the barrier with attributes referenced by *attr*. If *attr* is
 50248 NULL, the default barrier attributes shall be used; the effect is the same as passing the address of
 50249 a default barrier attributes object. The results are undefined if *pthread_barrier_init()* is called
 50250 when any thread is blocked on the barrier (that is, has not returned from the
 50251 *pthread_barrier_wait()* call). The results are undefined if a barrier is used without first being
 50252 initialized. The results are undefined if *pthread_barrier_init()* is called specifying an already
 50253 initialized barrier.

50254 The *count* argument specifies the number of threads that must call *pthread_barrier_wait()* before
 50255 any of them successfully return from the call. The value specified by *count* must be greater than
 50256 zero.

50257 If the *pthread_barrier_init()* function fails, the barrier shall not be initialized and the contents of
 50258 *barrier* are undefined.

50259 Only the object referenced by *barrier* may be used for performing synchronization. The result of
 50260 referring to copies of that object in calls to *pthread_barrier_destroy()* or *pthread_barrier_wait()* is
 50261 undefined.

50262 **RETURN VALUE**

50263 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 50264 be returned to indicate the error.

50265 **ERRORS**

50266 The *pthread_barrier_init()* function shall fail if:

50267 [EAGAIN] The system lacks the necessary resources to initialize another barrier.

50268 [EINVAL] The value specified by *count* is equal to zero.

50269 [ENOMEM] Insufficient memory exists to initialize the barrier.

50270 These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *barrier* argument to *pthread_barrier_destroy()* does not refer to an initialized barrier object, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *attr* argument to *pthread_barrier_init()* does not refer to an initialized barrier attributes object, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *barrier* argument to *pthread_barrier_destroy()* or *pthread_barrier_init()* refers to a barrier that is in use (for example, in a *pthread_barrier_wait()* call) by another thread, or detects that the value specified by the *barrier* argument to *pthread_barrier_init()* refers to an already initialized barrier object, it is recommended that the function should fail and report an [EBUSY] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_barrier_wait()

XBD <pthread.h>

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7

The *pthread_barrier_destroy()* and *pthread_barrier_init()* functions are moved from the Barriers option to the Base.

The [EINVAL] error for an uninitialized barrier object and an uninitialized barrier attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for a barrier that is in use or an already initialized barrier object is removed; this condition results in undefined behavior.

pthread_barrier_wait()*System Interfaces*50301 **NAME**

50302 pthread_barrier_wait — synchronize at a barrier

50303 **SYNOPSIS**

50304 #include <pthread.h>

50305 int pthread_barrier_wait(pthread_barrier_t *barrier);

50306 **DESCRIPTION**

50307 The *pthread_barrier_wait()* function shall synchronize participating threads at the barrier
 50308 referenced by *barrier*. The calling thread shall block until the required number of threads have
 50309 called *pthread_barrier_wait()* specifying the barrier.

50310 When the required number of threads have called *pthread_barrier_wait()* specifying the barrier,
 50311 the constant PTHREAD_BARRIER_SERIAL_THREAD shall be returned to one unspecified
 50312 thread and zero shall be returned to each of the remaining threads. At this point, the barrier shall
 50313 be reset to the state it had as a result of the most recent *pthread_barrier_init()* function that
 50314 referenced it.

50315 The constant PTHREAD_BARRIER_SERIAL_THREAD is defined in <pthread.h> and its value
 50316 shall be distinct from any other value returned by *pthread_barrier_wait()*.

50317 The results are undefined if this function is called with an uninitialized barrier.

50318 If a signal is delivered to a thread blocked on a barrier, upon return from the signal handler the
 50319 thread shall resume waiting at the barrier if the barrier wait has not completed (that is, if the
 50320 required number of threads have not arrived at the barrier during the execution of the signal
 50321 handler); otherwise, the thread shall continue as normal from the completed barrier wait. Until
 50322 the thread in the signal handler returns from it, it is unspecified whether other threads may
 50323 proceed past the barrier once they have all reached it.

50324 A thread that has blocked on a barrier shall not prevent any unblocked thread that is eligible to
 50325 use the same processing resources from eventually making forward progress in its execution.
 50326 Eligibility for processing resources shall be determined by the scheduling policy.

50327 **RETURN VALUE**

50328 Upon successful completion, the *pthread_barrier_wait()* function shall return
 50329 PTHREAD_BARRIER_SERIAL_THREAD for a single (arbitrary) thread synchronized at the
 50330 barrier and zero for each of the other threads. Otherwise, an error number shall be returned to
 50331 indicate the error.

50332 **ERRORS**

50333 This function shall not return an error code of [EINTR].

50334 **EXAMPLES**

50335 None.

50336 **APPLICATION USAGE**

50337 Applications using this function may be subject to priority inversion, as discussed in XBD
 50338 [Section 3.285](#) (on page 79).

50339 **RATIONALE**

50340 If an implementation detects that the value specified by the *barrier* argument to
 50341 *pthread_barrier_wait()* does not refer to an initialized barrier object, it is recommended that the
 50342 function should fail and report an [EINVAL] error.

50343 **FUTURE DIRECTIONS**

50344 None.

50345 **SEE ALSO**50346 [*pthread_barrier_destroy\(\)*](#)50347 XBD [Section 3.285](#) (on page 79), [Section 4.11](#) (on page 110), [**<pthread.h>**](#)50348 **CHANGE HISTORY**

50349 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

50350 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.50351 **Issue 7**50352 The *pthread_barrier_wait()* function is moved from the Barriers option to the Base.50353 The [EINVAL] error for an uninitialized barrier object is removed; this condition results in
50354 undefined behavior.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_barrierattr_destroy()*System Interfaces***50355 NAME**

50356 **pthread_barrierattr_destroy, pthread_barrierattr_init** — destroy and initialize the barrier
 50357 attributes object

50358 SYNOPSIS

50359 `#include <pthread.h>`
 50360 `int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);`
 50361 `int pthread_barrierattr_init(pthread_barrierattr_t *attr);`

50362 DESCRIPTION

50363 The *pthread_barrierattr_destroy()* function shall destroy a barrier attributes object. A destroyed
 50364 *attr* attributes object can be reinitialized using *pthread_barrierattr_init()*; the results of otherwise
 50365 referencing the object after it has been destroyed are undefined. An implementation may cause
 50366 *pthread_barrierattr_destroy()* to set the object referenced by *attr* to an invalid value.

50367 The *pthread_barrierattr_init()* function shall initialize a barrier attributes object *attr* with the
 50368 default value for all of the attributes defined by the implementation.

50369 If *pthread_barrierattr_init()* is called specifying an already initialized *attr* attributes object, the
 50370 results are undefined.

50371 After a barrier attributes object has been used to initialize one or more barriers, any function
 50372 affecting the attributes object (including destruction) shall not affect any previously initialized
 50373 barrier.

50374 The behavior is undefined if the value specified by the *attr* argument to
 50375 *pthread_barrierattr_destroy()* does not refer to an initialized barrier attributes object.

50376 RETURN VALUE

50377 If successful, the *pthread_barrierattr_destroy()* and *pthread_barrierattr_init()* functions shall return
 50378 zero; otherwise, an error number shall be returned to indicate the error.

50379 ERRORS

50380 The *pthread_barrierattr_init()* function shall fail if:

50381 [ENOMEM] Insufficient memory exists to initialize the barrier attributes object.

50382 These functions shall not return an error code of [EINTR].

50383 EXAMPLES

50384 None.

50385 APPLICATION USAGE

50386 None.

50387 RATIONALE

50388 If an implementation detects that the value specified by the *attr* argument to
 50389 *pthread_barrierattr_destroy()* does not refer to an initialized barrier attributes object, it is
 50390 recommended that the function should fail and report an [EINVAL] error.

50391 FUTURE DIRECTIONS

50392 None.

50393 SEE ALSO

50394 *pthread_barrierattr_getpshared()*

50395 XBD *<pthread.h>*

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

Issue 7

The `pthread_barrierattr_destroy()` and `pthread_barrierattr_init()` functions are moved from the Barriers option to the Base.

The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition results in undefined behavior.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_barrierattr_getpshared()

System Interfaces

50404 **NAME**

50405 pthread_barrierattr_getpshared, pthread_barrierattr_setpshared — get and set the process-
 50406 shared attribute of the barrier attributes object

50407 **SYNOPSIS**

```
50408 TSH      #include <pthread.h>
50409
50409      int pthread_barrierattr_getpshared(const pthread_barrierattr_t
50410          *restrict attr, int *restrict pshared);
50411
50411      int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
50412          int pshared);
```

50413 **DESCRIPTION**

50414 The *pthread_barrierattr_getpshared()* function shall obtain the value of the *process-shared* attribute
 50415 from the attributes object referenced by *attr*. The *pthread_barrierattr_setpshared()* function shall
 50416 set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

50417 The *process-shared* attribute is set to `PTHREAD_PROCESS_SHARED` to permit a barrier to be
 50418 operated upon by any thread that has access to the memory where the barrier is allocated. If the
 50419 *process-shared* attribute is `PTHREAD_PROCESS_PRIVATE`, the barrier shall only be operated
 50420 upon by threads created within the same process as the thread that initialized the barrier; if
 50421 threads of different processes attempt to operate on such a barrier, the behavior is undefined.
 50422 The default value of the attribute shall be `PTHREAD_PROCESS_PRIVATE`. Both constants
 50423 `PTHREAD_PROCESS_SHARED` and `PTHREAD_PROCESS_PRIVATE` are defined in
 50424 **<pthread.h>**.

50425 Additional attributes, their default values, and the names of the associated functions to get and
 50426 set those attribute values are implementation-defined.

50427 The behavior is undefined if the value specified by the *attr* argument to
 50428 *pthread_barrierattr_getpshared()* or *pthread_barrierattr_setpshared()* does not refer to an initialized
 50429 barrier attributes object.

50430 **RETURN VALUE**

50431 If successful, the *pthread_barrierattr_getpshared()* function shall return zero and store the value of
 50432 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,
 50433 an error number shall be returned to indicate the error.

50434 If successful, the *pthread_barrierattr_setpshared()* function shall return zero; otherwise, an error
 50435 number shall be returned to indicate the error.

50436 **ERRORS**

50437 The *pthread_barrierattr_setpshared()* function may fail if:

50438 [EINVAL] The new value specified for the *process-shared* attribute is not one of the legal
 50439 values `PTHREAD_PROCESS_SHARED` or `PTHREAD_PROCESS_PRIVATE`.

50440 These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

The *pthread_barrierattr_getpshared()* and *pthread_barrierattr_setpshared()* functions are part of the Thread Process-Shared Synchronization option and need not be provided on all implementations.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_barrierattr_getpshared()* or *pthread_barrierattr_setpshared()* does not refer to an initialized barrier attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_barrier_destroy(), *pthread_barrierattr_destroy()*

XBD <pthread.h>

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000

Issue 7

The *pthread_barrierattr_getpshared()* and *pthread_barrierattr_setpshared()* functions are moved from the Barriers option.

The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition results in undefined behavior.

pthread_barrierattr_init()*System Interfaces*50464 **NAME**

50465 pthread_barrierattr_init — initialize the barrier attributes object

50466 **SYNOPSIS**

50467 #include <pthread.h>

50468 int pthread_barrierattr_init(pthread_barrierattr_t *attr);

50469 **DESCRIPTION**50470 Refer to *pthread_barrierattr_destroy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

50471 **NAME**

50472 pthread_barrierattr_setpshared — set the process-shared attribute of the barrier attributes object

50473 **SYNOPSIS**

```
50474 TSH #include <pthread.h>
50475      int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
50476      int pshared);
```

50477 **DESCRIPTION**50478 Refer to [pthread_barrierattr_getpshared\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_cancel()

System Interfaces

50479 NAME

50480 pthread_cancel — cancel execution of a thread

50481 SYNOPSIS

50482 #include <pthread.h>

50483 int pthread_cancel(pthread_t thread);

50484 DESCRIPTION

50485 The *pthread_cancel()* function shall request that *thread* be canceled. The target thread's
 50486 cancelability state and type determines when the cancellation takes effect. When the cancellation
 50487 is acted on, the cancellation cleanup handlers for *thread* shall be called. When the last
 50488 cancellation cleanup handler returns, the thread-specific data destructor functions shall be called
 50489 for *thread*. When the last destructor function returns, *thread* shall be terminated.

50490 The cancellation processing in the target thread shall run asynchronously with respect to the
 50491 calling thread returning from *pthread_cancel()*.

50492 RETURN VALUE

50493 If successful, the *pthread_cancel()* function shall return zero; otherwise, an error number shall be
 50494 returned to indicate the error.

50495 ERRORS

50496 The *pthread_cancel()* function shall not return an error code of [EINTR].

50497 EXAMPLES

50498 None.

50499 APPLICATION USAGE

50500 None.

50501 RATIONALE

50502 Two alternative functions were considered for sending the cancellation notification to a thread.
 50503 One would be to define a new SIGCANCEL signal that had the cancellation semantics when
 50504 delivered; the other was to define the new *pthread_cancel()* function, which would trigger the
 50505 cancellation semantics.

50506 The advantage of a new signal was that so much of the delivery criteria were identical to that
 50507 used when trying to deliver a signal that making cancellation notification a signal was seen as
 50508 consistent. Indeed, many implementations implement cancellation using a special signal. On the
 50509 other hand, there would be no signal functions that could be used with this signal except
 50510 *pthread_kill()*, and the behavior of the delivered cancellation signal would be unlike any
 50511 previously existing defined signal.

50512 The benefits of a special function include the recognition that this signal would be defined
 50513 because of the similar delivery criteria and that this is the only common behavior between a
 50514 cancellation request and a signal. In addition, the cancellation delivery mechanism does not
 50515 have to be implemented as a signal. There are also strong, if not stronger, parallels with
 50516 language exception mechanisms than with signals that are potentially obscured if the delivery
 50517 mechanism is visibly closer to signals.

50518 In the end, it was considered that as there were so many exceptions to the use of the new signal
 50519 with existing signals functions it would be misleading. A special function has resolved this
 50520 problem. This function was carefully defined so that an implementation wishing to provide the
 50521 cancellation functions on top of signals could do so. The special function also means that
 50522 implementations are not obliged to implement cancellation with signals.

50523 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
 50524 that the function should fail and report an [ESRCH] error.

50525 **FUTURE DIRECTIONS**

50526 None.

50527 **SEE ALSO**50528 *pthread_exit()*, *pthread_cond_timedwait()*, *pthread_join()*, *pthread_setcancelstate()*50529 XBD **<pthread.h>**50530 **CHANGE HISTORY**

50531 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

50532 **Issue 6**50533 The *pthread_cancel()* function is marked as part of the Threads option.50534 **Issue 7**50535 The *pthread_cancel()* function is moved from the Threads option to the Base

50536 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_cleanup_pop()

System Interfaces

50537 **NAME**

50538 pthread_cleanup_pop, pthread_cleanup_push — establish cancellation handlers

50539 **SYNOPSIS**

50540 #include <pthread.h>

50541 void pthread_cleanup_pop(int execute);

50542 void pthread_cleanup_push(void (*routine)(void*), void *arg);

50543 **DESCRIPTION**50544 The *pthread_cleanup_pop()* function shall remove the routine at the top of the calling thread's
50545 cancellation cleanup stack and optionally invoke it (if *execute* is non-zero).50546 The *pthread_cleanup_push()* function shall push the specified cancellation cleanup handler *routine*
50547 onto the calling thread's cancellation cleanup stack. The cancellation cleanup handler shall be
50548 popped from the cancellation cleanup stack and invoked with the argument *arg* when:

- 50549
- The thread exits (that is, calls *pthread_exit()*).
 - The thread acts upon a cancellation request.
 - The thread calls *pthread_cleanup_pop()* with a non-zero *execute* argument.

50552 These functions may be implemented as macros. The application shall ensure that they appear
50553 as statements, and in pairs within the same lexical scope (that is, the *pthread_cleanup_push()*
50554 macro may be thought to expand to a token list whose first token is '{' with
50555 *pthread_cleanup_pop()* expanding to a token list whose last token is the corresponding '}').50556 The effect of calling *longjmp()* or *siglongjmp()* is undefined if there have been any calls to
50557 *pthread_cleanup_push()* or *pthread_cleanup_pop()* made without the matching call since the jump
50558 buffer was filled. The effect of calling *longjmp()* or *siglongjmp()* from inside a cancellation
50559 cleanup handler is also undefined unless the jump buffer was also filled in the cancellation
50560 cleanup handler.50561 The effect of the use of **return**, **break**, **continue**, and **goto** to prematurely leave a code block
50562 described by a pair of *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions calls is
50563 undefined.50564 **RETURN VALUE**50565 The *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions shall not return a value.50566 **ERRORS**

50567 No errors are defined.

50568 These functions shall not return an error code of [EINTR].

50569 **EXAMPLES**50570 The following is an example using thread primitives to implement a cancelable, writers-priority
50571 read-write lock:50572 typedef struct {
50573 pthread_mutex_t lock;
50574 pthread_cond_t rcond,
50575 wcond;
50576 int lock_count; /* < 0 .. Held by writer. */
50577 /* > 0 .. Held by lock_count readers. */
50578 /* = 0 .. Held by nobody. */
50579 int waiting_writers; /* Count of waiting writers. */
50580 } rwlock;

System Interfaces

pthread_cleanup_pop()

```

50581 void
50582 waiting_reader_cleanup(void *arg)
50583 {
50584     rwlock *l;
50585     l = (rwlock *) arg;
50586     pthread_mutex_unlock(&l->lock);
50587 }
50588 void
50589 lock_for_read(rwlock *l)
50590 {
50591     pthread_mutex_lock(&l->lock);
50592     pthread_cleanup_push(waiting_reader_cleanup, l);
50593     while ((l->lock_count < 0) && (l->waiting_writers != 0))
50594         pthread_cond_wait(&l->rcond, &l->lock);
50595     l->lock_count++;
50596     /*
50597      * Note the pthread_cleanup_pop executes
50598      * waiting_reader_cleanup.
50599      */
50600     pthread_cleanup_pop(1);
50601 }
50602 void
50603 release_read_lock(rwlock *l)
50604 {
50605     pthread_mutex_lock(&l->lock);
50606     if (--l->lock_count == 0)
50607         pthread_cond_signal(&l->wcond);
50608     pthread_mutex_unlock(&l->lock);
50609 }
50610 void
50611 waiting_writer_cleanup(void *arg)
50612 {
50613     rwlock *l;
50614     l = (rwlock *) arg;
50615     if ((l->waiting_writers == 0) && (l->lock_count >= 0)) {
50616         /*
50617          * This only happens if we have been canceled.
50618          */
50619         pthread_cond_broadcast(&l->wcond);
50620     }
50621     pthread_mutex_unlock(&l->lock);
50622 }
50623 void
50624 lock_for_write(rwlock *l)
50625 {
50626     pthread_mutex_lock(&l->lock);
50627     l->waiting_writers++;
50628     pthread_cleanup_push(waiting_writer_cleanup, l);
50629     while (l->lock_count != 0)

```

pthread_cleanup_pop()*System Interfaces*

```

50630         pthread_cond_wait(&l->wcond, &l->lock);
50631         l->lock_count = -1;
50632     /*
50633      * Note the pthread_cleanup_pop executes
50634      * waiting_writer_cleanup.
50635      */
50636     pthread_cleanup_pop(1);
50637 }

50638 void
50639 release_write_lock(rwlock *l)
50640 {
50641     pthread_mutex_lock(&l->lock);
50642     l->lock_count = 0;
50643     if (l->waiting_writers == 0)
50644         pthread_cond_broadcast(&l->rcond)
50645     else
50646         pthread_cond_signal(&l->wcond);
50647     pthread_mutex_unlock(&l->lock);
50648 }

50649 /*
50650  * This function is called to initialize the read/write lock.
50651  */
50652 void
50653 initialize_rwlock(rwlock *l)
50654 {
50655     pthread_mutex_init(&l->lock, pthread_mutexattr_default);
50656     pthread_cond_init(&l->wcond, pthread_condattr_default);
50657     pthread_cond_init(&l->rcond, pthread_condattr_default);
50658     l->lock_count = 0;
50659     l->waiting_writers = 0;
50660 }

50661 reader_thread()
50662 {
50663     lock_for_read(&lock);
50664     pthread_cleanup_push(release_read_lock, &lock);
50665     /*
50666      * Thread has read lock.
50667      */
50668     pthread_cleanup_pop(1);
50669 }

50670 writer_thread()
50671 {
50672     lock_for_write(&lock);
50673     pthread_cleanup_push(release_write_lock, &lock);
50674     /*
50675      * Thread has write lock.
50676      */
50677     pthread_cleanup_pop(1);
50678 }

```

APPLICATION USAGE

The two routines that push and pop cancellation cleanup handlers, *pthread_cleanup_push()* and *pthread_cleanup_pop()*, can be thought of as left and right-parentheses. They always need to be matched.

RATIONALE

The restriction that the two routines that push and pop cancellation cleanup handlers, *pthread_cleanup_push()* and *pthread_cleanup_pop()*, have to appear in the same lexical scope allows for efficient macro or compiler implementations and efficient storage management. A sample implementation of these routines as macros might look like this:

```
#define pthread_cleanup_push(rtn,arg) { \
    struct _pthread_handler_rec __cleanup_handler, *__head; \
    __cleanup_handler.rtn = rtn; \
    __cleanup_handler.arg = arg; \
    (void) pthread_getspecific(_pthread_handler_key, &__head); \
    __cleanup_handler.next = *__head; \
    *__head = &__cleanup_handler; \
}

#define pthread_cleanup_pop(ex) \
    *__head = __cleanup_handler.next; \
    if (ex) (*__cleanup_handler.rtn)(__cleanup_handler.arg); \
}
```

A more ambitious implementation of these routines might do even better by allowing the compiler to note that the cancellation cleanup handler is a constant and can be expanded inline.

This volume of POSIX.1-2008 currently leaves unspecified the effect of calling *longjmp()* from a signal handler executing in a POSIX System Interfaces function. If an implementation wants to allow this and give the programmer reasonable behavior, the *longjmp()* function has to call all cancellation cleanup handlers that have been pushed but not popped since the time *setjmp()* was called.

Consider a multi-threaded function called by a thread that uses signals. If a signal were delivered to a signal handler during the operation of *qsort()* and that handler were to call *longjmp()* (which, in turn, did *not* call the cancellation cleanup handlers) the helper threads created by the *qsort()* function would not be canceled. Instead, they would continue to execute and write into the argument array even though the array might have been popped off the stack.

Note that the specified cleanup handling mechanism is especially tied to the C language and, while the requirement for a uniform mechanism for expressing cleanup is language-independent, the mechanism used in other languages may be quite different. In addition, this mechanism is really only necessary due to the lack of a real exception mechanism in the C language, which would be the ideal solution.

There is no notion of a cancellation cleanup-safe function. If an application has no cancellation points in its signal handlers, blocks any signal whose handler may have cancellation points while calling async-unsafe functions, or disables cancellation while calling async-unsafe functions, all functions may be safely called from cancellation cleanup routines.

FUTURE DIRECTIONS

None.

pthread_cleanup_pop()

System Interfaces

50722 **SEE ALSO**50723 *pthread_cancel()*, *pthread_setcancelstate()*

50724 XBD <pthread.h>

50725 **CHANGE HISTORY**

50726 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

50727 **Issue 6**50728 The *pthread_cleanup_pop()* and *pthread_cleanup_push()* functions are marked as part of the
50729 Threads option.

50730 The APPLICATION USAGE section is added.

50731 The normative text is updated to avoid use of the term “must” for application requirements.

50732 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/88 is applied, updating the
50733 DESCRIPTION to describe the consequences of prematurely leaving a code block defined by the
50734 *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions.50735 **Issue 7**50736 The *pthread_cleanup_pop()* and *pthread_cleanup_push()* functions are moved from the Threads
50737 option to the Base.

NAME

pthread_cond_broadcast, pthread_cond_signal — broadcast or signal a condition

SYNOPSIS

```
#include <pthread.h>

int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_signal(pthread_cond_t *cond);
```

DESCRIPTION

These functions shall unblock threads blocked on a condition variable.

The *pthread_cond_broadcast()* function shall unblock all threads currently blocked on the specified condition variable *cond*.

The *pthread_cond_signal()* function shall unblock at least one of the threads that are blocked on the specified condition variable *cond* (if any threads are blocked on *cond*).

If more than one thread is blocked on a condition variable, the scheduling policy shall determine the order in which threads are unblocked. When each thread unblocked as a result of a *pthread_cond_broadcast()* or *pthread_cond_signal()* returns from its call to *pthread_cond_wait()* or *pthread_cond_timedwait()*, the thread shall own the mutex with which it called *pthread_cond_wait()* or *pthread_cond_timedwait()*. The thread(s) that are unblocked shall contend for the mutex according to the scheduling policy (if applicable), and as if each had called *pthread_mutex_lock()*.

The *pthread_cond_broadcast()* or *pthread_cond_signal()* functions may be called by a thread whether or not it currently owns the mutex that threads calling *pthread_cond_wait()* or *pthread_cond_timedwait()* have associated with the condition variable during their waits; however, if predictable scheduling behavior is required, then that mutex shall be locked by the thread calling *pthread_cond_broadcast()* or *pthread_cond_signal()*.

The *pthread_cond_broadcast()* and *pthread_cond_signal()* functions shall have no effect if there are no threads currently blocked on *cond*.

The behavior is undefined if the value specified by the *cond* argument to *pthread_cond_broadcast()* or *pthread_cond_signal()* does not refer to an initialized condition variable.

RETURN VALUE

If successful, the *pthread_cond_broadcast()* and *pthread_cond_signal()* functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

The *pthread_cond_broadcast()* function is used whenever the shared-variable state has been changed in a way that more than one thread can proceed with its task. Consider a single producer/multiple consumer problem, where the producer can insert multiple items on a list that is accessed one item at a time by the consumers. By calling the *pthread_cond_broadcast()* function, the producer would notify all consumers that might be waiting, and thereby the application would receive more throughput on a multi-processor. In addition, *pthread_cond_broadcast()* makes it easier to implement a read-write lock. The *pthread_cond_broadcast()* function is needed in order to wake up all waiting readers when a writer releases its lock. Finally, the two-phase commit algorithm can use this broadcast function to notify all clients of an impending transaction commit.

pthread_cond_broadcast()*System Interfaces*

It is not safe to use the *pthread_cond_signal()* function in a signal handler that is invoked asynchronously. Even if it were safe, there would still be a race between the test of the Boolean *pthread_cond_wait()* that could not be efficiently eliminated.

Mutexes and condition variables are thus not suitable for releasing a waiting thread by signaling from code running in a signal handler.

RATIONALE

If an implementation detects that the value specified by the *cond* argument to *pthread_cond_broadcast()* or *pthread_cond_signal()* does not refer to an initialized condition variable, it is recommended that the function should fail and report an [EINVAL] error.

Multiple Awakenings by Condition Signal

On a multi-processor, it may be impossible for an implementation of *pthread_cond_signal()* to avoid the unblocking of more than one thread blocked on a condition variable. For example, consider the following partial implementation of *pthread_cond_wait()* and *pthread_cond_signal()*, executed by two threads in the order given. One thread is trying to wait on the condition variable, another is concurrently executing *pthread_cond_signal()*, while a third thread is already waiting.

```
pthread_cond_wait(mutex, cond):
    value = cond->value; /* 1 */
    pthread_mutex_unlock(mutex); /* 2 */
    pthread_mutex_lock(cond->mutex); /* 10 */
    if (value == cond->value) { /* 11 */
        me->next_cond = cond->waiter;
        cond->waiter = me;
        pthread_mutex_unlock(cond->mutex);
        unable_to_run(me);
    } else
        pthread_mutex_unlock(cond->mutex); /* 12 */
    pthread_mutex_lock(mutex); /* 13 */

pthread_cond_signal(cond):
    pthread_mutex_lock(cond->mutex); /* 3 */
    cond->value++; /* 4 */
    if (cond->waiter) { /* 5 */
        sleeper = cond->waiter; /* 6 */
        cond->waiter = sleeper->next_cond; /* 7 */
        able_to_run(sleeper); /* 8 */
    }
    pthread_mutex_unlock(cond->mutex); /* 9 */
```

The effect is that more than one thread can return from its call to *pthread_cond_wait()* or *pthread_cond_timedwait()* as a result of one call to *pthread_cond_signal()*. This effect is called “spurious wakeup”. Note that the situation is self-correcting in that the number of threads that are so awakened is finite; for example, the next thread to call *pthread_cond_wait()* after the sequence of events above blocks.

While this problem could be resolved, the loss of efficiency for a fringe condition that occurs only rarely is unacceptable, especially given that one has to check the predicate associated with a condition variable anyway. Correcting this problem would unnecessarily reduce the degree of concurrency in this basic building block for all higher-level synchronization operations.

An added benefit of allowing spurious wakeups is that applications are forced to code a

50831 predicate-testing-loop around the condition wait. This also makes the application tolerate
 50832 superfluous condition broadcasts or signals on the same condition variable that may be coded in
 50833 some other part of the application. The resulting applications are thus more robust. Therefore,
 50834 POSIX.1-2008 explicitly documents that spurious wakeups may occur.

50835 **FUTURE DIRECTIONS**

50836 None.

50837 **SEE ALSO**

50838 [*pthread_cond_destroy\(\)*](#), [*pthread_cond_timedwait\(\)*](#)

50839 XBD [Section 4.11](#) (on page 110), [**<pthread.h>**](#)

50840 **CHANGE HISTORY**

50841 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

50842 **Issue 6**

50843 The *pthread_cond_broadcast()* and *pthread_cond_signal()* functions are marked as part of the
 50844 Threads option.

50845 The APPLICATION USAGE section is added.

50846 **Issue 7**

50847 The *pthread_cond_broadcast()* and *pthread_cond_signal()* functions are moved from the Threads
 50848 option to the Base.

50849 The [EINVAL] error for an uninitialized condition variable is removed; this condition results in
 50850 undefined behavior.

pthread_cond_destroy()

System Interfaces

50851 **NAME**

50852 pthread_cond_destroy, pthread_cond_init — destroy and initialize condition variables

50853 **SYNOPSIS**

```
50854 #include <pthread.h>
50855 int pthread_cond_destroy(pthread_cond_t *cond);
50856 int pthread_cond_init(pthread_cond_t *restrict cond,
50857     const pthread_condattr_t *restrict attr);
50858 pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

50859 **DESCRIPTION**

50860 The *pthread_cond_destroy()* function shall destroy the given condition variable specified by *cond*;
 50861 the object becomes, in effect, uninitialized. An implementation may cause *pthread_cond_destroy()*
 50862 to set the object referenced by *cond* to an invalid value. A destroyed condition variable object can
 50863 be reinitialized using *pthread_cond_init()*; the results of otherwise referencing the object after it
 50864 has been destroyed are undefined.

50865 It shall be safe to destroy an initialized condition variable upon which no threads are currently
 50866 blocked. Attempting to destroy a condition variable upon which other threads are currently
 50867 blocked results in undefined behavior.

50868 The *pthread_cond_init()* function shall initialize the condition variable referenced by *cond* with
 50869 attributes referenced by *attr*. If *attr* is NULL, the default condition variable attributes shall be
 50870 used; the effect is the same as passing the address of a default condition variable attributes
 50871 object. Upon successful initialization, the state of the condition variable shall become initialized.

50872 Only *cond* itself may be used for performing synchronization. The result of referring to copies of
 50873 *cond* in calls to *pthread_cond_wait()*, *pthread_cond_timedwait()*, *pthread_cond_signal()*,
 50874 *pthread_cond_broadcast()*, and *pthread_cond_destroy()* is undefined.

50875 Attempting to initialize an already initialized condition variable results in undefined behavior.

50876 In cases where default condition variable attributes are appropriate, the macro
 50877 PTHREAD_COND_INITIALIZER can be used to initialize condition variables that are statically
 50878 allocated. The effect shall be equivalent to dynamic initialization by a call to *pthread_cond_init()*
 50879 with parameter *attr* specified as NULL, except that no error checks are performed.

50880 The behavior is undefined if the value specified by the *cond* argument to *pthread_cond_destroy()*
 50881 does not refer to an initialized condition variable.

50882 The behavior is undefined if the value specified by the *attr* argument to *pthread_cond_init()* does
 50883 not refer to an initialized condition variable attributes object.

50884 **RETURN VALUE**

50885 If successful, the *pthread_cond_destroy()* and *pthread_cond_init()* functions shall return zero;
 50886 otherwise, an error number shall be returned to indicate the error.

50887 **ERRORS**

50888 The *pthread_cond_init()* function shall fail if:

50889 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize
 50890 another condition variable.

50891 [ENOMEM] Insufficient memory exists to initialize the condition variable.

50892 These functions shall not return an error code of [EINTR].

EXAMPLES

A condition variable can be destroyed immediately after all the threads that are blocked on it are awakened. For example, consider the following code:

```

struct list {
    pthread_mutex_t lm;
    ...
}

struct elt {
    key k;
    int busy;
    pthread_cond_t notbusy;
    ...
}

/* Find a list element and reserve it. */
struct elt *
list_find(struct list *lp, key k)
{
    struct elt *ep;

    pthread_mutex_lock(&lp->lm);
    while ((ep = find_elt(l, k) != NULL) && ep->busy)
        pthread_cond_wait(&ep->notbusy, &lp->lm);
    if (ep != NULL)
        ep->busy = 1;
    pthread_mutex_unlock(&lp->lm);
    return(ep);
}

delete_elt(struct list *lp, struct elt *ep)
{
    pthread_mutex_lock(&lp->lm);
    assert(ep->busy);
    ... remove ep from list ...
    ep->busy = 0; /* Paranoid. */
    (A) pthread_cond_broadcast(&ep->notbusy);
    pthread_mutex_unlock(&lp->lm);
    (B) pthread_cond_destroy(&rp->notbusy);
    free(ep);
}

```

In this example, the condition variable and its list element may be freed (line B) immediately after all threads waiting for it are awakened (line A), since the mutex and the code ensure that no other thread can touch the element to be deleted.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *cond* argument to *pthread_cond_destroy()* does not refer to an initialized condition variable, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *cond* argument to

pthread_cond_destroy()

System Interfaces

50940 *pthread_cond_destroy()* or *pthread_cond_init()* refers to a condition variable that is in use (for
 50941 example, in a *pthread_cond_wait()* call) by another thread, or detects that the value specified by
 50942 the *cond* argument to *pthread_cond_init()* refers to an already initialized condition variable, it is
 50943 recommended that the function should fail and report an [EBUSY] error.

50944 If an implementation detects that the value specified by the *attr* argument to *pthread_cond_init()*
 50945 does not refer to an initialized condition variable attributes object, it is recommended that the
 50946 function should fail and report an [EINVAL] error.

50947 See also *pthread_mutex_destroy()*.

FUTURE DIRECTIONS

50948 None.

SEE ALSO

50951 *pthread_cond_broadcast()*, *pthread_cond_timedwait()*, *pthread_mutex_destroy()*

50952 XBD <pthread.h>

CHANGE HISTORY

50953 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

50956 The *pthread_cond_destroy()* and *pthread_cond_init()* functions are marked as part of the Threads
 50957 option.

50958 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

50959 The **restrict** keyword is added to the *pthread_cond_init()* prototype for alignment with the
 50960 ISO/IEC 9899:1999 standard.

Issue 7

50962 The *pthread_cond_destroy()* and *pthread_cond_init()* functions are moved from the Threads option
 50963 to the Base.

50964 The [EINVAL] error for an uninitialized condition variable and an uninitialized condition
 50965 variable attributes object is removed; this condition results in undefined behavior.

50966 The [EBUSY] error for a condition variable already in use or an already initialized condition
 50967 variable is removed; this condition results in undefined behavior.

*System Interfaces***pthread_cond_signal()**50968 **NAME**

50969 pthread_cond_signal — signal a condition

50970 **SYNOPSIS**

50971 #include <pthread.h>

50972 int pthread_cond_signal(pthread_cond_t *cond);

50973 **DESCRIPTION**50974 Refer to *pthread_cond_broadcast()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_cond_timedwait()

System Interfaces

50975 **NAME**

50976 pthread_cond_timedwait, pthread_cond_wait — wait on a condition

50977 **SYNOPSIS**

```
50978 #include <pthread.h>
50979
50979 int pthread_cond_timedwait(pthread_cond_t *restrict cond,
50980 pthread_mutex_t *restrict mutex,
50981 const struct timespec *restrict abstime);
50982
50982 int pthread_cond_wait(pthread_cond_t *restrict cond,
50983 pthread_mutex_t *restrict mutex);
```

50984 **DESCRIPTION**

50985 The *pthread_cond_timedwait()* and *pthread_cond_wait()* functions shall block on a condition
 50986 variable. The application shall ensure that these functions are called with *mutex* locked by the
 50987 calling thread; otherwise, an error (for PTHREAD_MUTEX_ERRORCHECK and robust
 50988 mutexes) or undefined behavior (for other mutexes) results.

50989 These functions atomically release *mutex* and cause the calling thread to block on the condition
 50990 variable *cond*; atomically here means “atomically with respect to access by another thread to the
 50991 mutex and then the condition variable”. That is, if another thread is able to acquire the mutex
 50992 after the about-to-block thread has released it, then a subsequent call to *pthread_cond_broadcast()*
 50993 or *pthread_cond_signal()* in that thread shall behave as if it were issued after the about-to-block
 50994 thread has blocked.

50995 Upon successful return, the mutex shall have been locked and shall be owned by the calling
 50996 thread. If *mutex* is a robust mutex where an owner terminated while holding the lock and the
 50997 state is recoverable, the mutex shall be acquired even though the function returns an error code.

50998 When using condition variables there is always a Boolean predicate involving shared variables
 50999 associated with each condition wait that is true if the thread should proceed. Spurious wakeups
 51000 from the *pthread_cond_timedwait()* or *pthread_cond_wait()* functions may occur. Since the return
 51001 from *pthread_cond_timedwait()* or *pthread_cond_wait()* does not imply anything about the value of
 51002 this predicate, the predicate should be re-evaluated upon such return.

51003 When a thread waits on a condition variable, having specified a particular mutex to either the
 51004 *pthread_cond_timedwait()* or the *pthread_cond_wait()* operation, a dynamic binding is formed
 51005 between that mutex and condition variable that remains in effect as long as at least one thread is
 51006 blocked on the condition variable. During this time, the effect of an attempt by any thread to
 51007 wait on that condition variable using a different mutex is undefined. Once all waiting threads
 51008 have been unblocked (as by the *pthread_cond_broadcast()* operation), the next wait operation on
 51009 that condition variable shall form a new dynamic binding with the mutex specified by that wait
 51010 operation. Even though the dynamic binding between condition variable and mutex may be
 51011 removed or replaced between the time a thread is unblocked from a wait on the condition
 51012 variable and the time that it returns to the caller or begins cancellation cleanup, the unblocked
 51013 thread shall always re-acquire the mutex specified in the condition wait operation call from
 51014 which it is returning.

51015 A condition wait (whether timed or not) is a cancellation point. When the cancelability type of a
 51016 thread is set to PTHREAD_CANCEL_DEFERRED, a side-effect of acting upon a cancellation
 51017 request while in a condition wait is that the mutex is (in effect) re-acquired before calling the first
 51018 cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up
 51019 to the point of returning from the call to *pthread_cond_timedwait()* or *pthread_cond_wait()*, but at
 51020 that point notices the cancellation request and instead of returning to the caller of
 51021 *pthread_cond_timedwait()* or *pthread_cond_wait()*, starts the thread cancellation activities, which
 51022 includes calling cancellation cleanup handlers.

51023 A thread that has been unblocked because it has been canceled while blocked in a call to
 51024 *pthread_cond_timedwait()* or *pthread_cond_wait()* shall not consume any condition signal that may
 51025 be directed concurrently at the condition variable if there are other threads blocked on the
 51026 condition variable.

51027 The *pthread_cond_timedwait()* function shall be equivalent to *pthread_cond_wait()*, except that an
 51028 error is returned if the absolute time specified by *abstime* passes (that is, system time equals or
 51029 exceeds *abstime*) before the condition *cond* is signaled or broadcasted, or if the absolute time
 51030 specified by *abstime* has already been passed at the time of the call.

51031 The condition variable shall have a clock attribute which specifies the clock that shall be used to
 51032 measure the time specified by the *abstime* argument. When such timeouts occur,
 51033 *pthread_cond_timedwait()* shall nonetheless release and re-acquire the mutex referenced by *mutex*.
 51034 The *pthread_cond_timedwait()* function is also a cancellation point.

51035 If a signal is delivered to a thread waiting for a condition variable, upon return from the signal
 51036 handler the thread resumes waiting for the condition variable as if it was not interrupted, or it
 51037 shall return zero due to spurious wakeup.

51038 The behavior is undefined if the value specified by the *cond* or *mutex* argument to these
 51039 functions does not refer to an initialized condition variable or an initialized mutex object,
 51040 respectively.

51041 RETURN VALUE

51042 Except in the case of [ETIMEDOUT], all these error checks shall act as if they were performed
 51043 immediately at the beginning of processing for the function and shall cause an error return, in
 51044 effect, prior to modifying the state of the mutex specified by *mutex* or the condition variable
 51045 specified by *cond*.

51046 Upon successful completion, a value of zero shall be returned; otherwise, an error number shall
 51047 be returned to indicate the error.

51048 ERRORS

51049 These functions shall fail if:

51050 [ENOTRECOVERABLE]

51051 The state protected by the mutex is not recoverable.

51052 [EOWNERDEAD]

51053 The mutex is a robust mutex and the process containing the previous owning
 51054 thread terminated while holding the mutex lock. The mutex lock shall be
 51055 acquired by the calling thread and it is up to the new owner to make the state
 51056 consistent.

51057 [EPERM]

51058 The mutex type is PTHREAD_MUTEX_ERRORCHECK or the mutex is a
 robust mutex, and the current thread does not own the mutex.

51059 The *pthread_cond_timedwait()* function shall fail if:

51060 [ETIMEDOUT] The time specified by *abstime* to *pthread_cond_timedwait()* has passed.

51061 [EINVAL]

51062 The *abstime* argument specified a nanosecond value less than zero or greater
 than or equal to 1000 million.

51063 These functions may fail if:

51064 [EOWNERDEAD]

51065 The mutex is a robust mutex and the previous owning thread terminated
 51066 while holding the mutex lock. The mutex lock shall be acquired by the calling

pthread_cond_timedwait()*System Interfaces*

51067 thread and it is up to the new owner to make the state consistent.

51068 These functions shall not return an error code of [EINTR].

EXAMPLES

51070 None.

APPLICATION USAGE

51072 Applications that have assumed that non-zero return values are errors will need updating for
 51073 use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting
 51074 a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error
 51075 returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If
 51076 an application is supposed to work with normal and robust mutexes, it should check all return
 51077 values for error conditions and if necessary take appropriate action.

RATIONALE

51078 If an implementation detects that the value specified by the *cond* argument to
 51079 *pthread_cond_timedwait()* or *pthread_cond_wait()* does not refer to an initialized condition
 51080 variable, or detects that the value specified by the *mutex* argument to *pthread_cond_timedwait()* or
 51081 *pthread_cond_wait()* does not refer to an initialized mutex object, it is recommended that the
 51082 function should fail and report an [EINVAL] error.

Condition Wait Semantics

51084 It is important to note that when *pthread_cond_wait()* and *pthread_cond_timedwait()* return
 51085 without error, the associated predicate may still be false. Similarly, when
 51086 *pthread_cond_timedwait()* returns with the timeout error, the associated predicate may be true
 51087 due to an unavoidable race between the expiration of the timeout and the predicate state change.

51088 The application needs to recheck the predicate on any return because it cannot be sure there is
 51089 another thread waiting on the thread to handle the signal, and if there is not then the signal is
 51090 lost. The burden is on the application to check the predicate.

51091 Some implementations, particularly on a multi-processor, may sometimes cause multiple
 51092 threads to wake up when the condition variable is signaled simultaneously on different
 51093 processors.

51094 In general, whenever a condition wait returns, the thread has to re-evaluate the predicate
 51095 associated with the condition wait to determine whether it can safely proceed, should wait
 51096 again, or should declare a timeout. A return from the wait does not imply that the associated
 51097 predicate is either true or false.

51098 It is thus recommended that a condition wait be enclosed in the equivalent of a “while loop”
 51099 that checks the predicate.

Timed Wait Semantics

51101 An absolute time measure was chosen for specifying the timeout parameter for two reasons.
 51102 First, a relative time measure can be easily implemented on top of a function that specifies
 51103 absolute time, but there is a race condition associated with specifying an absolute timeout on top
 51104 of a function that specifies relative timeouts. For example, assume that *clock_gettime()* returns
 51105 the current time and *cond_relative_timed_wait()* uses relative timeouts:

```
51107 clock_gettime(CLOCK_REALTIME, &now)
51108 reltime = sleep_til_this_absolute_time - now;
51109 cond_relative_timed_wait(c, m, &reltime);
```

51110 If the thread is preempted between the first statement and the last statement, the thread blocks

for too long. Blocking, however, is irrelevant if an absolute timeout is used. An absolute timeout also need not be recomputed if it is used multiple times in a loop, such as that enclosing a condition wait.

For cases when the system clock is advanced discontinuously by an operator, it is expected that implementations process any timed wait expiring at an intervening time as if that time had actually occurred.

Cancellation and Condition Wait

A condition wait, whether timed or not, is a cancellation point. That is, the functions `pthread_cond_wait()` or `pthread_cond_timedwait()` are points where a pending (or concurrent) cancellation request is noticed. The reason for this is that an indefinite wait is possible at these points—whatever event is being waited for, even if the program is totally correct, might never occur; for example, some input data being awaited might never be sent. By making condition wait a cancellation point, the thread can be canceled and perform its cancellation cleanup handler even though it may be stuck in some indefinite wait.

A side-effect of acting on a cancellation request while a thread is blocked on a condition variable is to re-acquire the mutex before calling any of the cancellation cleanup handlers. This is done in order to ensure that the cancellation cleanup handler is executed in the same state as the critical code that lies both before and after the call to the condition wait function. This rule is also required when interfacing to POSIX threads from languages, such as Ada or C++, which may choose to map cancellation onto a language exception; this rule ensures that each exception handler guarding a critical section can always safely depend upon the fact that the associated mutex has already been locked regardless of exactly where within the critical section the exception was raised. Without this rule, there would not be a uniform rule that exception handlers could follow regarding the lock, and so coding would become very cumbersome.

Therefore, since *some* statement has to be made regarding the state of the lock when a cancellation is delivered during a wait, a definition has been chosen that makes application coding most convenient and error free.

When acting on a cancellation request while a thread is blocked on a condition variable, the implementation is required to ensure that the thread does not consume any condition signals directed at that condition variable if there are any other threads waiting on that condition variable. This rule is specified in order to avoid deadlock conditions that could occur if these two independent requests (one acting on a thread and the other acting on the condition variable) were not processed independently.

Performance of Mutexes and Condition Variables

Mutexes are expected to be locked only for a few instructions. This practice is almost automatically enforced by the desire of programmers to avoid long serial regions of execution (which would reduce total effective parallelism).

When using mutexes and condition variables, one tries to ensure that the usual case is to lock the mutex, access shared data, and unlock the mutex. Waiting on a condition variable should be a relatively rare situation. For example, when implementing a read-write lock, code that acquires a read-lock typically needs only to increment the count of readers (under mutual-exclusion) and return. The calling thread would actually wait on the condition variable only when there is already an active writer. So the efficiency of a synchronization operation is bounded by the cost of mutex lock/unlock and not by condition wait. Note that in the usual case there is no context switch.

This is not to say that the efficiency of condition waiting is unimportant. Since there needs to be

pthread_cond_timedwait()*System Interfaces*

at least one context switch per Ada rendezvous, the efficiency of waiting on a condition variable is important. The cost of waiting on a condition variable should be little more than the minimal cost for a context switch plus the time to unlock and lock the mutex.

Features of Mutexes and Condition Variables

It had been suggested that the mutex acquisition and release be decoupled from condition wait. This was rejected because it is the combined nature of the operation that, in fact, facilitates realtime implementations. Those implementations can atomically move a high-priority thread between the condition variable and the mutex in a manner that is transparent to the caller. This can prevent extra context switches and provide more deterministic acquisition of a mutex when the waiting thread is signaled. Thus, fairness and priority issues can be dealt with directly by the scheduling discipline. Furthermore, the current condition wait operation matches existing practice.

Scheduling Behavior of Mutexes and Condition Variables

Synchronization primitives that attempt to interfere with scheduling policy by specifying an ordering rule are considered undesirable. Threads waiting on mutexes and condition variables are selected to proceed in an order dependent upon the scheduling policy rather than in some fixed order (for example, FIFO or priority). Thus, the scheduling policy determines which thread(s) are awakened and allowed to proceed.

Timed Condition Wait

The `pthread_cond_timedwait()` function allows an application to give up waiting for a particular condition after a given amount of time. An example of its use follows:

```
(void) pthread_mutex_lock(&t.mn);
    t.waiters++;
    clock_gettime(CLOCK_REALTIME, &ts);
    ts.tv_sec += 5;
    rc = 0;
    while (! mypredicate(&t) && rc == 0)
        rc = pthread_cond_timedwait(&t.cond, &t.mn, &ts);
    t.waiters--;
    if (rc == 0) setmystate(&t);
(void) pthread_mutex_unlock(&t.mn);
```

By making the timeout parameter absolute, it does not need to be recomputed each time the program checks its blocking predicate. If the timeout was relative, it would have to be recomputed before each call. This would be especially difficult since such code would need to take into account the possibility of extra wakeups that result from extra broadcasts or signals on the condition variable that occur before either the predicate is true or the timeout is due.

FUTURE DIRECTIONS

None.

SEE ALSO

[*pthread_cond_broadcast\(\)*](#)

XBD [Section 4.11](#) (on page 110), [*<pthread.h>*](#)

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_cond_timedwait()* and *pthread_cond_wait()* functions are marked as part of the Threads option.

The Open Group Corrigendum U021/9 is applied, correcting the prototype for the *pthread_cond_wait()* function.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for the Clock Selection option.

The ERRORS section has an additional case for [EPERM] in response to IEEE PASC Interpretation 1003.1c #28.

The **restrict** keyword is added to the *pthread_cond_timedwait()* and *pthread_cond_wait()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/89 is applied, updating the DESCRIPTION for consistency with the *pthread_cond_destroy()* function that states it is safe to destroy an initialized condition variable upon which no threads are currently blocked.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/90 is applied, updating words in the DESCRIPTION from “the cancelability enable state” to “the cancelability type”.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/91 is applied, updating the ERRORS section to remove the error case related to *abstime* from the *pthread_cond_wait()* function, and to make the error case related to *abstime* mandatory for *pthread_cond_timedwait()* for consistency with other functions.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/92 is applied, adding a new paragraph to the RATIONALE section stating that an application should check the predicate on any return from this function.

Issue 7

SD5-XSH-ERN-44 is applied, changing the definition of the “shall fail” case of the [EINVAL] error.

Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_cond_timedwait()* and *pthread_cond_wait()* functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable or uninitialized mutex object is removed; this condition results in undefined behavior”

The [EPERM] error is revised and moved to the “shall fail” list of error conditions for the *pthread_cond_timedwait()* function.

The DESCRIPTION is updated to clarify the behavior when *mutex* is a robust mutex.

The ERRORS section is updated to include “shall fail” cases for PTHREAD_MUTEX_ERRORCHECK mutexes.

The DESCRIPTION is rewritten to clarify that undefined behavior occurs only for mutexes where the [EPERM] error is not mandated.

pthread_condattr_destroy()

System Interfaces

51238 **NAME**

51239 pthread_condattr_destroy, pthread_condattr_init — destroy and initialize the condition variable
 51240 attributes object

51241 **SYNOPSIS**

```
51242 #include <pthread.h>

51243 int pthread_condattr_destroy(pthread_condattr_t *attr);
51244 int pthread_condattr_init(pthread_condattr_t *attr);
```

51245 **DESCRIPTION**

51246 The *pthread_condattr_destroy()* function shall destroy a condition variable attributes object; the
 51247 object becomes, in effect, uninitialized. An implementation may cause *pthread_condattr_destroy()*
 51248 to set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be
 51249 reinitialized using *pthread_condattr_init()*; the results of otherwise referencing the object after it
 51250 has been destroyed are undefined.

51251 The *pthread_condattr_init()* function shall initialize a condition variable attributes object *attr* with
 51252 the default value for all of the attributes defined by the implementation.

51253 Results are undefined if *pthread_condattr_init()* is called specifying an already initialized *attr*
 51254 attributes object.

51255 After a condition variable attributes object has been used to initialize one or more condition
 51256 variables, any function affecting the attributes object (including destruction) shall not affect any
 51257 previously initialized condition variables.

51258 This volume of POSIX.1-2008 requires two attributes, the *clock* attribute and the *process-shared*
 51259 attribute.

51260 Additional attributes, their default values, and the names of the associated functions to get and
 51261 set those attribute values are implementation-defined.

51262 The behavior is undefined if the value specified by the *attr* argument to
 51263 *pthread_condattr_destroy()* does not refer to an initialized condition variable attributes object.

51264 **RETURN VALUE**

51265 If successful, the *pthread_condattr_destroy()* and *pthread_condattr_init()* functions shall return
 51266 zero; otherwise, an error number shall be returned to indicate the error.

51267 **ERRORS**

51268 The *pthread_condattr_init()* function shall fail if:

51269 [ENOMEM] Insufficient memory exists to initialize the condition variable attributes object.

51270 These functions shall not return an error code of [EINTR].

51271 **EXAMPLES**

51272 None.

51273 **APPLICATION USAGE**

51274 None.

51275 **RATIONALE**

51276 A *process-shared* attribute has been defined for condition variables for the same reason it has been
 51277 defined for mutexes.

51278 If an implementation detects that the value specified by the *attr* argument to
 51279 *pthread_condattr_destroy()* does not refer to an initialized condition variable attributes object, it is
 51280 recommended that the function should fail and report an [EINVAL] error.

51281 See also *pthread_attr_destroy()* and *pthread_mutex_destroy()*.

51282 FUTURE DIRECTIONS

51283 None.

51284 SEE ALSO

51285 *pthread_attr_destroy()*, *pthread_cond_destroy()*, *pthread_condattr_getpshared()*, *pthread_create()*,
51286 *pthread_mutex_destroy()*

51287 XBD **<pthread.h>**

51288 CHANGE HISTORY

51289 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51290 Issue 6

51291 The *pthread_condattr_destroy()* and *pthread_condattr_init()* functions are marked as part of the
51292 Threads option.

51293 Issue 7

51294 The *pthread_condattr_destroy()* and *pthread_condattr_init()* functions are moved from the Threads
51295 option to the Base.

51296 The [EINVAL] error for an uninitialized condition variable attributes object is removed; this
51297 condition results in undefined behavior.

pthread_condattr_getclock()

System Interfaces

51298 **NAME**

51299 pthread_condattr_getclock, pthread_condattr_setclock — get and set the clock selection
 51300 condition variable attribute

51301 **SYNOPSIS**

```
51302 #include <pthread.h>

51303 int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
51304 clockid_t *restrict clock_id);
51305 int pthread_condattr_setclock(pthread_condattr_t *attr,
51306 clockid_t clock_id);
```

51307 **DESCRIPTION**

51308 The *pthread_condattr_getclock()* function shall obtain the value of the *clock* attribute from the
 51309 attributes object referenced by *attr*.

51310 The *pthread_condattr_setclock()* function shall set the *clock* attribute in an initialized attributes
 51311 object referenced by *attr*. If *pthread_condattr_setclock()* is called with a *clock_id* argument that
 51312 refers to a CPU-time clock, the call shall fail.

51313 The *clock* attribute is the clock ID of the clock that shall be used to measure the timeout service of
 51314 *pthread_cond_timedwait()*. The default value of the *clock* attribute shall refer to the system clock.

51315 The behavior is undefined if the value specified by the *attr* argument to
 51316 *pthread_condattr_getclock()* or *pthread_condattr_setclock()* does not refer to an initialized condition
 51317 variable attributes object.

51318 **RETURN VALUE**

51319 If successful, the *pthread_condattr_getclock()* function shall return zero and store the value of the
 51320 clock attribute of *attr* into the object referenced by the *clock_id* argument. Otherwise, an error
 51321 number shall be returned to indicate the error.

51322 If successful, the *pthread_condattr_setclock()* function shall return zero; otherwise, an error
 51323 number shall be returned to indicate the error.

51324 **ERRORS**

51325 The *pthread_condattr_setclock()* function may fail if:

51326 [EINVAL] The value specified by *clock_id* does not refer to a known clock, or is a CPU-
 51327 time clock.

51328 These functions shall not return an error code of [EINTR].

51329 **EXAMPLES**

51330 None.

51331 **APPLICATION USAGE**

51332 None.

51333 **RATIONALE**

51334 If an implementation detects that the value specified by the *attr* argument to
 51335 *pthread_condattr_getclock()* or *pthread_condattr_setclock()* does not refer to an initialized condition
 51336 variable attributes object, it is recommended that the function should fail and report an
 51337 [EINVAL] error.

51338 **FUTURE DIRECTIONS**

51339 None.

51340 **SEE ALSO**

51341 *pthread_cond_destroy()*, *pthread_cond_timedwait()*, *pthread_condattr_destroy()*,
51342 *pthread_condattr_getpshared()*, *pthread_create()*, *pthread_mutex_destroy()*

51343 XBD <pthread.h>

51344 **CHANGE HISTORY**

51345 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

51346 **Issue 7**

51347 The *pthread_condattr_getclock()* and *pthread_condattr_setclock()* functions are moved from the
51348 Clock Selection option to the Base.

51349 The [EINVAL] error for an uninitialized condition variable attributes object is removed; this
51350 condition results in undefined behavior.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_condattr_getpshared()

System Interfaces

51351 **NAME**

51352 pthread_condattr_getpshared, pthread_condattr_setpshared — get and set the process-shared
 51353 condition variable attributes

51354 **SYNOPSIS**

```
51355 TSH      #include <pthread.h>
51356
51357      int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
51358                                     int *restrict pshared);
51359      int pthread_condattr_setpshared(pthread_condattr_t *attr,
51360                                     int pshared);
```

51360 **DESCRIPTION**

51361 The *pthread_condattr_getpshared()* function shall obtain the value of the *process-shared* attribute
 51362 from the attributes object referenced by *attr*.

51363 The *pthread_condattr_setpshared()* function shall set the *process-shared* attribute in an initialized
 51364 attributes object referenced by *attr*.

51365 The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a condition
 51366 variable to be operated upon by any thread that has access to the memory where the condition
 51367 variable is allocated, even if the condition variable is allocated in memory that is shared by
 51368 multiple processes. If the *process-shared* attribute is PTHREAD_PROCESS_PRIVATE, the
 51369 condition variable shall only be operated upon by threads created within the same process as the
 51370 thread that initialized the condition variable; if threads of differing processes attempt to operate
 51371 on such a condition variable, the behavior is undefined. The default value of the attribute is
 51372 PTHREAD_PROCESS_PRIVATE.

51373 The behavior is undefined if the value specified by the *attr* argument to
 51374 *pthread_condattr_getpshared()* or *pthread_condattr_setpshared()* does not refer to an initialized
 51375 condition variable attributes object.

51376 **RETURN VALUE**

51377 If successful, the *pthread_condattr_setpshared()* function shall return zero; otherwise, an error
 51378 number shall be returned to indicate the error.

51379 If successful, the *pthread_condattr_getpshared()* function shall return zero and store the value of
 51380 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,
 51381 an error number shall be returned to indicate the error.

51382 **ERRORS**

51383 The *pthread_condattr_setpshared()* function may fail if:

51384 [EINVAL] The new value specified for the attribute is outside the range of legal values
 51385 for that attribute.

51386 These functions shall not return an error code of [EINTR].

51387 EXAMPLES

51388 None.

51389 APPLICATION USAGE

51390 None.

51391 RATIONALE

51392 If an implementation detects that the value specified by the *attr* argument to
 51393 *pthread_condattr_getpshared()* or *pthread_condattr_setpshared()* does not refer to an initialized
 51394 condition variable attributes object, it is recommended that the function should fail and report
 51395 an [EINVAL] error.

51396 FUTURE DIRECTIONS

51397 None.

51398 SEE ALSO

51399 *pthread_create()*, *pthread_cond_destroy()*, *pthread_condattr_destroy()*, *pthread_mutex_destroy()*

51400 XBD <pthread.h>

51401 CHANGE HISTORY

51402 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51403 Issue 6

51404 The *pthread_condattr_getpshared()* and *pthread_condattr_setpshared()* functions are marked as part
 51405 of the Threads and Thread Process-Shared Synchronization options.

51406 The **restrict** keyword is added to the *pthread_condattr_getpshared()* prototype for alignment with
 51407 the ISO/IEC 9899: 1999 standard.

51408 Issue 7

51409 The *pthread_condattr_getpshared()* and *pthread_condattr_setpshared()* functions are moved from the
 51410 Threads option.

51411 The [EINVAL] error for an uninitialized condition variable attributes object is removed; this
 51412 condition results in undefined behavior.

pthread_condattr_init()*System Interfaces*51413 **NAME**

51414 pthread_condattr_init — initialize the condition variable attributes object

51415 **SYNOPSIS**

51416 #include <pthread.h>

51417 int pthread_condattr_init(pthread_condattr_t *attr);

51418 **DESCRIPTION**51419 Refer to *pthread_condattr_destroy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread_condattr_setclock()**51420 **NAME**

51421 pthread_condattr_setclock — set the clock selection condition variable attribute

51422 **SYNOPSIS**

51423 #include <pthread.h>

51424 int pthread_condattr_setclock(pthread_condattr_t *attr,
51425 clockid_t clock_id);51426 **DESCRIPTION**51427 Refer to *pthread_condattr_getclock()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_condattr_setpshared()*System Interfaces*51428 **NAME**

51429 pthread_condattr_setpshared — set the process-shared condition variable attribute

51430 **SYNOPSIS**

```
51431 TSH #include <pthread.h>
51432      int pthread_condattr_setpshared(pthread_condattr_t *attr,
51433      int pshared);
```

51434 **DESCRIPTION**51435 Refer to *pthread_condattr_getpshared()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

pthread_create — thread creation

SYNOPSIS

```
#include <pthread.h>

int pthread_create(pthread_t *restrict thread,
    const pthread_attr_t *restrict attr,
    void *(*start_routine) (void*), void *restrict arg);
```

DESCRIPTION

The *pthread_create()* function shall create a new thread, with attributes specified by *attr*, within a process. If *attr* is NULL, the default attributes shall be used. If the attributes specified by *attr* are modified later, the thread's attributes shall not be affected. Upon successful completion, *pthread_create()* shall store the ID of the created thread in the location referenced by *thread*.

The thread is created executing *start_routine* with *arg* as its sole argument. If the *start_routine* returns, the effect shall be as if there was an implicit call to *pthread_exit()* using the return value of *start_routine* as the exit status. Note that the thread in which *main()* was originally invoked differs from this. When it returns from *main()*, the effect shall be as if there was an implicit call to *exit()* using the return value of *main()* as the exit status.

The signal state of the new thread shall be initialized as follows:

- The signal mask shall be inherited from the creating thread.
- The set of signals pending for the new thread shall be empty.

XSI The alternate stack shall not be inherited.

The floating-point environment shall be inherited from the creating thread.

If *pthread_create()* fails, no new thread is created and the contents of the location referenced by *thread* are undefined.

TCT If `_POSIX_THREAD_CPUTIME` is defined, the new thread shall have a CPU-time clock accessible, and the initial value of this clock shall be set to zero.

The behavior is undefined if the value specified by the *attr* argument to *pthread_create()* does not refer to an initialized thread attributes object.

RETURN VALUE

If successful, the *pthread_create()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_create()* function shall fail if:

[EAGAIN] The system lacked the necessary resources to create another thread, or the system-imposed limit on the total number of threads in a process (`PTHREAD_THREADS_MAX`) would be exceeded.

[EPERM] The caller does not have appropriate privileges to set the required scheduling parameters or scheduling policy.

The *pthread_create()* function shall not return an error code of `[EINTR]`.

pthread_create()*System Interfaces***EXAMPLES**

None.

APPLICATION USAGE

There is no requirement on the implementation that the ID of the created thread be available before the newly created thread starts executing. The calling thread can obtain the ID of the created thread through the return value of the *pthread_create()* function, and the newly created thread can obtain its ID by a call to *pthread_self()*.

RATIONALE

A suggested alternative to *pthread_create()* would be to define two separate operations: create and start. Some applications would find such behavior more natural. Ada, in particular, separates the “creation” of a task from its “activation”.

Splitting the operation was rejected by the standard developers for many reasons:

- The number of calls required to start a thread would increase from one to two and thus place an additional burden on applications that do not require the additional synchronization. The second call, however, could be avoided by the additional complication of a start-up state attribute.
- An extra state would be introduced: “created but not started”. This would require the standard to specify the behavior of the thread operations when the target has not yet started executing.
- For those applications that require such behavior, it is possible to simulate the two separate steps with the facilities that are currently provided. The *start_routine()* can synchronize by waiting on a condition variable that is signaled by the start operation.

An Ada implementor can choose to create the thread at either of two points in the Ada program: when the task object is created, or when the task is activated (generally at a “begin”). If the first approach is adopted, the *start_routine()* needs to wait on a condition variable to receive the order to begin “activation”. The second approach requires no such condition variable or extra synchronization. In either approach, a separate Ada task control block would need to be created when the task object is created to hold rendezvous queues, and so on.

An extension of the preceding model would be to allow the state of the thread to be modified between the create and start. This would allow the thread attributes object to be eliminated. This has been rejected because:

- All state in the thread attributes object has to be able to be set for the thread. This would require the definition of functions to modify thread attributes. There would be no reduction in the number of function calls required to set up the thread. In fact, for an application that creates all threads using identical attributes, the number of function calls required to set up the threads would be dramatically increased. Use of a thread attributes object permits the application to make one set of attribute setting function calls. Otherwise, the set of attribute setting function calls needs to be made for each thread creation.
- Depending on the implementation architecture, functions to set thread state would require kernel calls, or for other implementation reasons would not be able to be implemented as macros, thereby increasing the cost of thread creation.
- The ability for applications to segregate threads by class would be lost.

Another suggested alternative uses a model similar to that for process creation, such as “thread fork”. The fork semantics would provide more flexibility and the “create” function can be implemented simply by doing a thread fork followed immediately by a call to the desired “start

routine” for the thread. This alternative has these problems:

- For many implementations, the entire stack of the calling thread would need to be duplicated, since in many architectures there is no way to determine the size of the calling frame.
- Efficiency is reduced since at least some part of the stack has to be copied, even though in most cases the thread never needs the copied context, since it merely calls the desired start routine.

If an implementation detects that the value specified by the *attr* argument to *pthread_create()* does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

fork(), *pthread_exit()*, *pthread_join()*

XBD Section 4.11 (on page 110), **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_create()* function is marked as part of the Threads option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EPERM] mandatory error condition is added.

The thread CPU-time clock semantics are added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_create()* prototype for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION is updated to make it explicit that the floating-point environment is inherited from the creating thread.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/44 is applied, adding text that the alternate stack is not inherited.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/93 is applied, updating the ERRORS section to remove the mandatory [EINVAL] error (“The value specified by *attr* is invalid”), and adding the optional [EINVAL] error (“The attributes specified by *attr* are invalid”).

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/94 is applied, adding the APPLICATION USAGE section.

Issue 7

The *pthread_create()* function is moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_detach()

System Interfaces

51559 **NAME**

51560 pthread_detach — detach a thread

51561 **SYNOPSIS**

51562 #include <pthread.h>

51563 int pthread_detach(pthread_t thread);

51564 **DESCRIPTION**

51565 The *pthread_detach()* function shall indicate to the implementation that storage for the thread
 51566 *thread* can be reclaimed when that thread terminates. If *thread* has not terminated,
 51567 *pthread_detach()* shall not cause it to terminate.

51568 The behavior is undefined if the value specified by the *thread* argument to *pthread_detach()* does
 51569 not refer to a joinable thread.

51570 **RETURN VALUE**

51571 If the call succeeds, *pthread_detach()* shall return 0; otherwise, an error number shall be returned
 51572 to indicate the error.

51573 **ERRORS**51574 The *pthread_detach()* function shall not return an error code of [EINTR].51575 **EXAMPLES**

51576 None.

51577 **APPLICATION USAGE**

51578 None.

51579 **RATIONALE**

51580 The *pthread_join()* or *pthread_detach()* functions should eventually be called for every thread that
 51581 is created so that storage associated with the thread may be reclaimed.

51582 It has been suggested that a “detach” function is not necessary; the *detachstate* thread creation
 51583 attribute is sufficient, since a thread need never be dynamically detached. However, need arises
 51584 in at least two cases:

51585 1. In a cancellation handler for a *pthread_join()* it is nearly essential to have a
 51586 *pthread_detach()* function in order to detach the thread on which *pthread_join()* was
 51587 waiting. Without it, it would be necessary to have the handler do another *pthread_join()* to
 51588 attempt to detach the thread, which would both delay the cancellation processing for an
 51589 unbounded period and introduce a new call to *pthread_join()*, which might itself need a
 51590 cancellation handler. A dynamic detach is nearly essential in this case.

51591 2. In order to detach the “initial thread” (as may be desirable in processes that set up server
 51592 threads).

51593 If an implementation detects that the value specified by the *thread* argument to *pthread_detach()*
 51594 does not refer to a joinable thread, it is recommended that the function should fail and report an
 51595 [EINVAL] error.

51596 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
 51597 that the function should fail and report an [ESRCH] error.

51598 **FUTURE DIRECTIONS**

51599 None.

51600 **SEE ALSO**51601 [*pthread_join\(\)*](#)51602 XBD [**<pthread.h>**](#)51603 **CHANGE HISTORY**

51604 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51605 **Issue 6**51606 The *pthread_detach()* function is marked as part of the Threads option.51607 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/95 is applied, updating the ERRORS
51608 section so that the [EINVAL] and [ESRCH] error cases become optional.51609 **Issue 7**51610 The *pthread_detach()* function is moved from the Threads option to the Base.

51611 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

51612 The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined
51613 behavior.

pthread_equal()

System Interfaces

51614 **NAME**

51615 pthread_equal — compare thread IDs

51616 **SYNOPSIS**

51617 #include <pthread.h>

51618 int pthread_equal(pthread_t t1, pthread_t t2);

51619 **DESCRIPTION**51620 This function shall compare the thread IDs *t1* and *t2*.51621 **RETURN VALUE**51622 The *pthread_equal()* function shall return a non-zero value if *t1* and *t2* are equal; otherwise, zero
51623 shall be returned.51624 If either *t1* or *t2* are not valid thread IDs, the behavior is undefined.51625 **ERRORS**

51626 No errors are defined.

51627 The *pthread_equal()* function shall not return an error code of [EINTR].51628 **EXAMPLES**

51629 None.

51630 **APPLICATION USAGE**

51631 None.

51632 **RATIONALE**

51633 Implementations may choose to define a thread ID as a structure. This allows additional
 51634 flexibility and robustness over using an **int**. For example, a thread ID could include a sequence
 51635 number that allows detection of “dangling IDs” (copies of a thread ID that has been detached).
 51636 Since the C language does not support comparison on structure types, the *pthread_equal()*
 51637 function is provided to compare thread IDs.

51638 **FUTURE DIRECTIONS**

51639 None.

51640 **SEE ALSO**51641 *pthread_create()*, *pthread_self()*

51642 XBD <pthread.h>

51643 **CHANGE HISTORY**

51644 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51645 **Issue 6**51646 The *pthread_equal()* function is marked as part of the Threads option.51647 **Issue 7**51648 The *pthread_equal()* function is moved from the Threads option to the Base.

NAME

pthread_exit — thread termination

SYNOPSIS

#include <pthread.h>

void pthread_exit(void *value_ptr);

DESCRIPTION

The *pthread_exit()* function shall terminate the calling thread and make the value *value_ptr* available to any successful join with the terminating thread. Any cancellation cleanup handlers that have been pushed and not yet popped shall be popped in the reverse order that they were pushed and then executed. After all cancellation cleanup handlers have been executed, if the thread has any thread-specific data, appropriate destructor functions shall be called in an unspecified order. Thread termination does not release any application visible process resources, including, but not limited to, mutexes and file descriptors, nor does it perform any process-level cleanup actions, including, but not limited to, calling any *atexit()* routines that may exist.

An implicit call to *pthread_exit()* is made when a thread other than the thread in which *main()* was first invoked returns from the start routine that was used to create it. The function's return value shall serve as the thread's exit status.

The behavior of *pthread_exit()* is undefined if called from a cancellation cleanup handler or destructor function that was invoked as a result of either an implicit or explicit call to *pthread_exit()*.

After a thread has terminated, the result of access to local (auto) variables of the thread is undefined. Thus, references to local variables of the exiting thread should not be used for the *pthread_exit()* *value_ptr* parameter value.

The process shall exit with an exit status of 0 after the last thread has been terminated. The behavior shall be as if the implementation called *exit()* with a zero argument at thread termination time.

RETURN VALUE

The *pthread_exit()* function cannot return to its caller.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The normal mechanism by which a thread terminates is to return from the routine that was specified in the *pthread_create()* call that started it. The *pthread_exit()* function provides the capability for a thread to terminate without requiring a return from the start routine of that thread, thereby providing a function analogous to *exit()*.

Regardless of the method of thread termination, any cancellation cleanup handlers that have been pushed and not yet popped are executed, and the destructors for any existing thread-specific data are executed. This volume of POSIX.1-2008 requires that cancellation cleanup handlers be popped and called in order. After all cancellation cleanup handlers have been executed, thread-specific data destructors are called, in an unspecified order, for each item of thread-specific data that exists in the thread. This ordering is necessary because cancellation cleanup handlers may rely on thread-specific data.

pthread_exit()*System Interfaces*

51695 As the meaning of the status is determined by the application (except when the thread has been
51696 canceled, in which case it is PTHREAD_CANCELED), the implementation has no idea what an
51697 illegal status value is, which is why no address error checking is done.

51698 FUTURE DIRECTIONS

51699 None.

51700 SEE ALSO

51701 *exit()*, *pthread_create()*, *pthread_join()*

51702 XBD **<pthread.h>**

51703 CHANGE HISTORY

51704 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51705 Issue 6

51706 The *pthread_exit()* function is marked as part of the Threads option.

51707 Issue 7

51708 The *pthread_exit()* function is moved from the Threads option to the Base.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

51709 NAME

51710 pthread_getconcurrency, pthread_setconcurrency — get and set the level of concurrency

51711 SYNOPSIS

```
51712 OB XSI #include <pthread.h>
51713
51713 int pthread_getconcurrency(void);
51714 int pthread_setconcurrency(int new_level);
```

51715 DESCRIPTION

51716 Unbound threads in a process may or may not be required to be simultaneously active. By
 51717 default, the threads implementation ensures that a sufficient number of threads are active so that
 51718 the process can continue to make progress. While this conserves system resources, it may not
 51719 produce the most effective level of concurrency.

51720 The *pthread_setconcurrency()* function allows an application to inform the threads
 51721 implementation of its desired concurrency level, *new_level*. The actual level of concurrency
 51722 provided by the implementation as a result of this function call is unspecified.

51723 If *new_level* is zero, it causes the implementation to maintain the concurrency level at its
 51724 discretion as if *pthread_setconcurrency()* had never been called.

51725 The *pthread_getconcurrency()* function shall return the value set by a previous call to the
 51726 *pthread_setconcurrency()* function. If the *pthread_setconcurrency()* function was not previously
 51727 called, this function shall return zero to indicate that the implementation is maintaining the
 51728 concurrency level.

51729 A call to *pthread_setconcurrency()* shall inform the implementation of its desired concurrency
 51730 level. The implementation shall use this as a hint, not a requirement.

51731 If an implementation does not support multiplexing of user threads on top of several kernel-
 51732 scheduled entities, the *pthread_setconcurrency()* and *pthread_getconcurrency()* functions are
 51733 provided for source code compatibility but they shall have no effect when called. To maintain
 51734 the function semantics, the *new_level* parameter is saved when *pthread_setconcurrency()* is called
 51735 so that a subsequent call to *pthread_getconcurrency()* shall return the same value.

51736 RETURN VALUE

51737 If successful, the *pthread_setconcurrency()* function shall return zero; otherwise, an error number
 51738 shall be returned to indicate the error.

51739 The *pthread_getconcurrency()* function shall always return the concurrency level set by a previous
 51740 call to *pthread_setconcurrency()*. If the *pthread_setconcurrency()* function has never been called,
 51741 *pthread_getconcurrency()* shall return zero.

51742 ERRORS

51743 The *pthread_setconcurrency()* function shall fail if:

51744 [EINVAL] The value specified by *new_level* is negative.

51745 [EAGAIN] The value specified by *new_level* would cause a system resource to be
 51746 exceeded.

51747 The *pthread_setconcurrency()* function shall not return an error code of [EINTR].

pthread_getconcurrency()*System Interfaces*51748 **EXAMPLES**

51749 None.

51750 **APPLICATION USAGE**

51751 Application developers should note that an implementation can always ignore any calls to
51752 *pthread_setconcurrency()* and return a constant for *pthread_getconcurrency()*. For this reason, it is
51753 not recommended that portable applications use this function.

51754 **RATIONALE**

51755 None.

51756 **FUTURE DIRECTIONS**

51757 These functions may be removed in a future version.

51758 **SEE ALSO**51759 XBD [<pthread.h>](#)51760 **CHANGE HISTORY**

51761 First released in Issue 5.

51762 **Issue 7**

51763 SD5-XSH-ERN-184 is applied.

51764 The *pthread_getconcurrency()* and *pthread_setconcurrency()* functions are marked obsolescent.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

51765 **NAME**

51766 pthread_getcpuclockid — access a thread CPU-time clock (**ADVANCED REALTIME**
 51767 **THREADS**)

51768 **SYNOPSIS**

```
51769 TCT #include <pthread.h>
51770 #include <time.h>
51771 int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
```

51772 **DESCRIPTION**

51773 The *pthread_getcpuclockid()* function shall return in *clock_id* the clock ID of the CPU-time clock of
 51774 the thread specified by *thread_id*, if the thread specified by *thread_id* exists.

51775 **RETURN VALUE**

51776 Upon successful completion, *pthread_getcpuclockid()* shall return zero; otherwise, an error
 51777 number shall be returned to indicate the error.

51778 **ERRORS**

51779 No errors are defined.

51780 **EXAMPLES**

51781 None.

51782 **APPLICATION USAGE**

51783 The *pthread_getcpuclockid()* function is part of the Thread CPU-Time Clocks option and need not
 51784 be provided on all implementations.

51785 **RATIONALE**

51786 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
 51787 that the function should fail and report an [ESRCH] error.

51788 **FUTURE DIRECTIONS**

51789 None.

51790 **SEE ALSO**

51791 *clock_getcpuclockid()*, *clock_getres()*, *timer_create()*

51792 XBD **<pthread.h>**, **<time.h>**

51793 **CHANGE HISTORY**

51794 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

51795 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

51796 **Issue 7**

51797 The *pthread_getcpuclockid()* function is moved from the Threads option.

51798 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

pthread_getschedparam()

System Interfaces

51799 **NAME**

51800 pthread_getschedparam, pthread_setschedparam — dynamic thread scheduling parameters
 51801 access (**REALTIME THREADS**)

51802 **SYNOPSIS**

```
51803 TPS #include <pthread.h>
51804
51804 int pthread_getschedparam(pthread_t thread, int *restrict policy,
51805 struct sched_param *restrict param);
51806
51806 int pthread_setschedparam(pthread_t thread, int policy,
51807 const struct sched_param *param);
```

51808 **DESCRIPTION**

51809 The *pthread_getschedparam()* and *pthread_setschedparam()* functions shall, respectively, get and set
 51810 the scheduling policy and parameters of individual threads within a multi-threaded process to
 51811 be retrieved and set. For SCHED_FIFO and SCHED_RR, the only required member of the
 51812 **sched_param** structure is the priority *sched_priority*. For SCHED_OTHER, the affected
 51813 scheduling parameters are implementation-defined.

51814 The *pthread_getschedparam()* function shall retrieve the scheduling policy and scheduling
 51815 parameters for the thread whose thread ID is given by *thread* and shall store those values in
 51816 *policy* and *param*, respectively. The priority value returned from *pthread_getschedparam()* shall be
 51817 the value specified by the most recent *pthread_setschedparam()*, *pthread_setschedprio()*, or
 51818 *pthread_create()* call affecting the target thread. It shall not reflect any temporary adjustments to
 51819 its priority as a result of any priority inheritance or ceiling functions. The *pthread_setschedparam()*
 51820 function shall set the scheduling policy and associated scheduling parameters for the thread
 51821 whose thread ID is given by *thread* to the policy and associated parameters provided in *policy*
 51822 and *param*, respectively.

51823 The *policy* parameter may have the value SCHED_OTHER, SCHED_FIFO, or SCHED_RR. The
 51824 scheduling parameters for the SCHED_OTHER policy are implementation-defined. The
 51825 SCHED_FIFO and SCHED_RR policies shall have a single scheduling parameter, *priority*.

51826 TSP If _POSIX_THREAD_SPORADIC_SERVER is defined, then the *policy* argument may have the
 51827 value SCHED_SPORADIC, with the exception for the *pthread_setschedparam()* function that if the
 51828 scheduling policy was not SCHED_SPORADIC at the time of the call, it is implementation-
 51829 defined whether the function is supported; in other words, the implementation need not allow
 51830 the application to dynamically change the scheduling policy to SCHED_SPORADIC. The
 51831 sporadic server scheduling policy has the associated parameters *sched_ss_low_priority*,
 51832 *sched_ss_repl_period*, *sched_ss_init_budget*, *sched_priority*, and *sched_ss_max_repl*. The specified
 51833 *sched_ss_repl_period* shall be greater than or equal to the specified *sched_ss_init_budget* for the
 51834 function to succeed; if it is not, then the function shall fail. The value of *sched_ss_max_repl* shall
 51835 be within the inclusive range [1,SS_REPL_MAX] for the function to succeed; if not, the function
 51836 shall fail. It is unspecified whether the *sched_ss_repl_period* and *sched_ss_init_budget* values are
 51837 stored as provided by this function or are rounded to align with the resolution of the clock being
 51838 used.

51839 If the *pthread_setschedparam()* function fails, the scheduling parameters shall not be changed for
 51840 the target thread.

51841 **RETURN VALUE**

51842 If successful, the *pthread_getschedparam()* and *pthread_setschedparam()* functions shall return zero;
 51843 otherwise, an error number shall be returned to indicate the error.

51844 ERRORS

51845 The *pthread_setschedparam()* function may fail if:

51846	[EINVAL]	The value specified by <i>policy</i> or one of the scheduling parameters associated with the scheduling policy <i>policy</i> is invalid.
51847		
51848	[ENOTSUP]	An attempt was made to set the policy or scheduling parameters to an unsupported value.
51849		
51850	TSP [ENOTSUP]	An attempt was made to dynamically change the scheduling policy to SCHED_SPORADIC, and the implementation does not support this change.
51851		
51852	[EPERM]	The caller does not have appropriate privileges to set either the scheduling parameters or the scheduling policy of the specified thread.
51853		
51854	[EPERM]	The implementation does not allow the application to modify one of the parameters to the value specified.
51855		

51856 These functions shall not return an error code of [EINTR].

51857 EXAMPLES

51858 None.

51859 APPLICATION USAGE

51860 None.

51861 RATIONALE

51862 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
51863 that the function should fail and report an [ESRCH] error.

51864 FUTURE DIRECTIONS

51865 None.

51866 SEE ALSO

51867 *pthread_setschedprio()*, *sched_getparam()*, *sched_getscheduler()*

51868 XBD **<pthread.h>**, **<sched.h>**

51869 CHANGE HISTORY

51870 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51871 Issue 6

51872 The *pthread_getschedparam()* and *pthread_setschedparam()* functions are marked as part of the
51873 Threads and Thread Execution Scheduling options.

51874 The [ENOSYS] error condition has been removed as stubs need not be provided if an
51875 implementation does not support the Thread Execution Scheduling option.

51876 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the
51877 *pthread_setschedparam()* function so that its second argument is of type **int**.

51878 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

51879 The **restrict** keyword is added to the *pthread_getschedparam()* prototype for alignment with the
51880 ISO/IEC 9899:1999 standard.

51881 The Open Group Corrigendum U047/1 is applied.

51882 IEEE PASC Interpretation 1003.1 #96 is applied, noting that priority values can also be set by a
51883 call to the *pthread_setschedprio()* function.

pthread_getschedparam()*System Interfaces*51884 **Issue 7**

51885 The *pthread_getschedparam()* and *pthread_setschedparam()* functions are moved from the Threads
51886 option.

51887 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements
51888 for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

51889 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

pthread_getspecific, pthread_setspecific — thread-specific data management

SYNOPSIS

```
#include <pthread.h>
```

```
void *pthread_getspecific(pthread_key_t key);
```

```
int pthread_setspecific(pthread_key_t key, const void *value);
```

DESCRIPTION

The *pthread_getspecific()* function shall return the value currently bound to the specified *key* on behalf of the calling thread.

The *pthread_setspecific()* function shall associate a thread-specific *value* with a *key* obtained via a previous call to *pthread_key_create()*. Different threads may bind different values to the same key. These values are typically pointers to blocks of dynamically allocated memory that have been reserved for use by the calling thread.

The effect of calling *pthread_getspecific()* or *pthread_setspecific()* with a *key* value not obtained from *pthread_key_create()* or after *key* has been deleted with *pthread_key_delete()* is undefined.

Both *pthread_getspecific()* and *pthread_setspecific()* may be called from a thread-specific data destructor function. A call to *pthread_getspecific()* for the thread-specific data key being destroyed shall return the value NULL, unless the value is changed (after the destructor starts) by a call to *pthread_setspecific()*. Calling *pthread_setspecific()* from a thread-specific data destructor routine may result either in lost storage (after at least PTHREAD_DESTRUCTOR_ITERATIONS attempts at destruction) or in an infinite loop.

Both functions may be implemented as macros.

RETURN VALUE

The *pthread_getspecific()* function shall return the thread-specific data value associated with the given *key*. If no thread-specific data value is associated with *key*, then the value NULL shall be returned.

If successful, the *pthread_setspecific()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

No errors are returned from *pthread_getspecific()*.

The *pthread_setspecific()* function shall fail if:

[ENOMEM] Insufficient memory exists to associate the non-NULL value with the key.

The *pthread_setspecific()* function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Performance and ease-of-use of *pthread_getspecific()* are critical for functions that rely on maintaining state in thread-specific data. Since no errors are required to be detected by it, and since the only error that could be detected is the use of an invalid key, the function to *pthread_getspecific()* has been designed to favor speed and simplicity over error reporting.

If an implementation detects that the value specified by the *key* argument to *pthread_setspecific()* does not refer to a key value obtained from *pthread_key_create()* or refers to a key that has been

pthread_getspecific()

System Interfaces

51934 deleted with *pthread_key_delete()*, it is recommended that the function should fail and report an
 51935 [EINVAL] error.

51936 **FUTURE DIRECTIONS**

51937 None.

51938 **SEE ALSO**

51939 *pthread_key_create()*

51940 XBD <pthread.h>

51941 **CHANGE HISTORY**

51942 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51943 **Issue 6**

51944 The *pthread_getspecific()* and *pthread_setspecific()* functions are marked as part of the Threads
 51945 option.

51946 IEEE PASC Interpretation 1003.1c #3 (Part 6) is applied, updating the DESCRIPTION.

51947 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/96 is applied, updating the ERRORS
 51948 section so that the [ENOMEM] error case is changed from “to associate the value with the key”
 51949 to “to associate the non-NULL value with the key”.

51950 **Issue 7**

51951 Austin Group Interpretation 1003.1-2001 #063 is applied, updating the ERRORS section.

51952 The *pthread_getspecific()* and *pthread_setspecific()* functions are moved from the Threads option to
 51953 the Base.

51954 The [EINVAL] error for a key value not obtained from *pthread_key_create()* or a key deleted with
 51955 *pthread_key_delete()* is removed; this condition results in undefined behavior.

NAME

pthread_join — wait for thread termination

SYNOPSIS

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **value_ptr);
```

DESCRIPTION

The *pthread_join()* function shall suspend execution of the calling thread until the target *thread* terminates, unless the target *thread* has already terminated. On return from a successful *pthread_join()* call with a non-NULL *value_ptr* argument, the value passed to *pthread_exit()* by the terminating thread shall be made available in the location referenced by *value_ptr*. When a *pthread_join()* returns successfully, the target thread has been terminated. The results of multiple simultaneous calls to *pthread_join()* specifying the same target thread are undefined. If the thread calling *pthread_join()* is canceled, then the target thread shall not be detached.

It is unspecified whether a thread that has exited but remains unjoined counts against {PTHREAD_THREADS_MAX}.

The behavior is undefined if the value specified by the *thread* argument to *pthread_join()* does not refer to a joinable thread.

The behavior is undefined if the value specified by the *thread* argument to *pthread_join()* refers to the calling thread.

RETURN VALUE

If successful, the *pthread_join()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_join()* function may fail if:

[EDEADLK] A deadlock was detected.

The *pthread_join()* function shall not return an error code of [EINTR].

EXAMPLES

An example of thread creation and deletion follows:

```
typedef struct {
    int *ar;
    long n;
} subarray;

void *
incr(void *arg)
{
    long i;

    for (i = 0; i < ((subarray *)arg)->n; i++)
        ((subarray *)arg)->ar[i]++;
}

int main(void)
{
    int ar[1000000];
    pthread_t th1, th2;
    subarray sb1, sb2;
```

pthread_join()*System Interfaces*

```

52000         sb1.ar = &ar[0];
52001         sb1.n  = 500000;
52002         (void) pthread_create(&th1, NULL, incer, &sb1);

52003         sb2.ar = &ar[500000];
52004         sb2.n  = 500000;
52005         (void) pthread_create(&th2, NULL, incer, &sb2);

52006         (void) pthread_join(th1, NULL);
52007         (void) pthread_join(th2, NULL);
52008         return 0;
52009     }

```

APPLICATION USAGE

None.

RATIONALE

The *pthread_join()* function is a convenience that has proven useful in multi-threaded applications. It is true that a programmer could simulate this function if it were not provided by passing extra state as part of the argument to the *start_routine()*. The terminating thread would set a flag to indicate termination and broadcast a condition that is part of that state; a joining thread would wait on that condition variable. While such a technique would allow a thread to wait on more complex conditions (for example, waiting for multiple threads to terminate), waiting on individual thread termination is considered widely useful. Also, including the *pthread_join()* function in no way precludes a programmer from coding such complex waits. Thus, while not a primitive, including *pthread_join()* in this volume of POSIX.1-2008 was considered valuable.

The *pthread_join()* function provides a simple mechanism allowing an application to wait for a thread to terminate. After the thread terminates, the application may then choose to clean up resources that were used by the thread. For instance, after *pthread_join()* returns, any application-provided stack storage could be reclaimed.

The *pthread_join()* or *pthread_detach()* function should eventually be called for every thread that is created with the *detachstate* attribute set to *PTHREAD_CREATE_JOINABLE* so that storage associated with the thread may be reclaimed.

The interaction between *pthread_join()* and cancellation is well-defined for the following reasons:

- The *pthread_join()* function, like all other non-async-cancel-safe functions, can only be called with deferred cancelability type.
- Cancellation cannot occur in the disabled cancelability state.

Thus, only the default cancelability state need be considered. As specified, either the *pthread_join()* call is canceled, or it succeeds, but not both. The difference is obvious to the application, since either a cancellation handler is run or *pthread_join()* returns. There are no race conditions since *pthread_join()* was called in the deferred cancelability state.

If an implementation detects that the value specified by the *thread* argument to *pthread_join()* does not refer to a joinable thread, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *thread* argument to *pthread_join()* refers to the calling thread, it is recommended that the function should fail and report an [EDEADLK] error.

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended

52045 that the function should fail and report an [ESRCH] error.

52046 **FUTURE DIRECTIONS**

52047 None.

52048 **SEE ALSO**

52049 *pthread_create()*, *wait()*

52050 XBD Section 4.11 (on page 110), *<pthread.h>*

52051 **CHANGE HISTORY**

52052 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

52053 **Issue 6**

52054 The *pthread_join()* function is marked as part of the Threads option.

52055 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/97 is applied, updating the ERRORS
52056 section so that the [EINVAL] error is made optional and the words “the implementation has
52057 detected” are removed from it.

52058 **Issue 7**

52059 The *pthread_join()* function is moved from the Threads option to the Base.

52060 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

52061 The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined
52062 behavior.

52063 The [EDEADLK] error for the calling thread is removed; this condition results in undefined
52064 behavior.

pthread_key_create()

System Interfaces

52065 NAME

52066 pthread_key_create — thread-specific data key creation

52067 SYNOPSIS

52068 #include <pthread.h>

52069 int pthread_key_create(pthread_key_t *key, void (*destructor)(void*));

52070 DESCRIPTION

52071 The *pthread_key_create()* function shall create a thread-specific data key visible to all threads in
 52072 the process. Key values provided by *pthread_key_create()* are opaque objects used to locate
 52073 thread-specific data. Although the same key value may be used by different threads, the values
 52074 bound to the key by *pthread_setspecific()* are maintained on a per-thread basis and persist for the
 52075 life of the calling thread.

52076 Upon key creation, the value NULL shall be associated with the new key in all active threads.
 52077 Upon thread creation, the value NULL shall be associated with all defined keys in the new
 52078 thread.

52079 An optional destructor function may be associated with each key value. At thread exit, if a key
 52080 value has a non-NULL destructor pointer, and the thread has a non-NULL value associated with
 52081 that key, the value of the key is set to NULL, and then the function pointed to is called with the
 52082 previously associated value as its sole argument. The order of destructor calls is unspecified if
 52083 more than one destructor exists for a thread when it exits.

52084 If, after all the destructors have been called for all non-NULL values with associated destructors,
 52085 there are still some non-NULL values with associated destructors, then the process is repeated.
 52086 If, after at least {PTHREAD_DESTRUCTOR_ITERATIONS} iterations of destructor calls for
 52087 outstanding non-NULL values, there are still some non-NULL values with associated
 52088 destructors, implementations may stop calling destructors, or they may continue calling
 52089 destructors until no non-NULL values with associated destructors exist, even though this might
 52090 result in an infinite loop.

52091 RETURN VALUE

52092 If successful, the *pthread_key_create()* function shall store the newly created key value at *key and
 52093 shall return zero. Otherwise, an error number shall be returned to indicate the error.

52094 ERRORS52095 The *pthread_key_create()* function shall fail if:

52096 [EAGAIN] The system lacked the necessary resources to create another thread-specific
 52097 data key, or the system-imposed limit on the total number of keys per process
 52098 {PTHREAD_KEYS_MAX} has been exceeded.

52099 [ENOMEM] Insufficient memory exists to create the key.

52100 The *pthread_key_create()* function shall not return an error code of [EINTR].

52101 EXAMPLES

52102 The following example demonstrates a function that initializes a thread-specific data key when it
 52103 is first called, and associates a thread-specific object with each calling thread, initializing this
 52104 object when necessary.

```
52105 static pthread_key_t key;
52106 static pthread_once_t key_once = PTHREAD_ONCE_INIT;

52107 static void
52108 make_key()
52109 {
```



```

52110         (void) pthread_key_create(&key, NULL);
52111     }

52112     func()
52113     {
52114         void *ptr;

52115         (void) pthread_once(&key_once, make_key);
52116         if ((ptr = pthread_getspecific(key)) == NULL) {
52117             ptr = malloc(OBJECT_SIZE);
52118             ...
52119             (void) pthread_setspecific(key, ptr);
52120         }
52121         ...
52122     }

```

52123 Note that the key has to be initialized before *pthread_getspecific()* or *pthread_setspecific()* can be
52124 used. The *pthread_key_create()* call could either be explicitly made in a module initialization
52125 routine, or it can be done implicitly by the first call to a module as in this example. Any attempt
52126 to use the key before it is initialized is a programming error, making the code below incorrect.

```

52127     static pthread_key_t key;

52128     func()
52129     {
52130         void *ptr;

52131         /* KEY NOT INITIALIZED!!! THIS WON'T WORK!!! */
52132         if ((ptr = pthread_getspecific(key)) == NULL &&
52133             pthread_setspecific(key, NULL) != 0) {
52134             pthread_key_create(&key, NULL);
52135             ...
52136         }
52137     }

```

52138 APPLICATION USAGE

52139 None.

52140 RATIONALE

52141 Destructor Functions

52142 Normally, the value bound to a key on behalf of a particular thread is a pointer to storage
52143 allocated dynamically on behalf of the calling thread. The destructor functions specified with
52144 *pthread_key_create()* are intended to be used to free this storage when the thread exits. Thread
52145 cancellation cleanup handlers cannot be used for this purpose because thread-specific data may
52146 persist outside the lexical scope in which the cancellation cleanup handlers operate.

52147 If the value associated with a key needs to be updated during the lifetime of the thread, it may
52148 be necessary to release the storage associated with the old value before the new value is bound.
52149 Although the *pthread_setspecific()* function could do this automatically, this feature is not needed
52150 often enough to justify the added complexity. Instead, the programmer is responsible for freeing
52151 the stale storage:

```

52152     pthread_getspecific(key, &old);
52153     new = allocate();
52154     destructor(old);

```

pthread_key_create()*System Interfaces*

52155 pthread_setspecific(key, new);

52156 **Note:** The above example could leak storage if run with asynchronous cancellation enabled. No such
52157 problems occur in the default cancellation state if no cancellation points occur between the get
52158 and set.

52159 There is no notion of a destructor-safe function. If an application does not call *pthread_exit()*
52160 from a signal handler, or if it blocks any signal whose handler may call *pthread_exit()* while
52161 calling async-unsafe functions, all functions may be safely called from destructors.

52162 **Non-Idempotent Data Key Creation**

52163 There were requests to make *pthread_key_create()* idempotent with respect to a given *key* address
52164 parameter. This would allow applications to call *pthread_key_create()* multiple times for a given
52165 *key* address and be guaranteed that only one key would be created. Doing so would require the
52166 key value to be previously initialized (possibly at compile time) to a known null value and
52167 would require that implicit mutual-exclusion be performed based on the address and contents of
52168 the *key* parameter in order to guarantee that exactly one key would be created.

52169 Unfortunately, the implicit mutual-exclusion would not be limited to only *pthread_key_create()*.
52170 On many implementations, implicit mutual-exclusion would also have to be performed by
52171 *pthread_getspecific()* and *pthread_setspecific()* in order to guard against using incompletely stored
52172 or not-yet-visible key values. This could significantly increase the cost of important operations,
52173 particularly *pthread_getspecific()*.

52174 Thus, this proposal was rejected. The *pthread_key_create()* function performs no implicit
52175 synchronization. It is the responsibility of the programmer to ensure that it is called exactly once
52176 per key before use of the key. Several straightforward mechanisms can already be used to
52177 accomplish this, including calling explicit module initialization functions, using mutexes, and
52178 using *pthread_once()*. This places no significant burden on the programmer, introduces no
52179 possibly confusing *ad hoc* implicit synchronization mechanism, and potentially allows
52180 commonly used thread-specific data operations to be more efficient.

52181 **FUTURE DIRECTIONS**

52182 None.

52183 **SEE ALSO**

52184 *pthread_getspecific()*, *pthread_key_delete()*

52185 XBD <pthread.h>

52186 **CHANGE HISTORY**

52187 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

52188 **Issue 6**

52189 The *pthread_key_create()* function is marked as part of the Threads option.

52190 IEEE PASC Interpretation 1003.1c #8 is applied, updating the DESCRIPTION.

52191 **Issue 7**

52192 The *pthread_key_create()* function is moved from the Threads option to the Base.

NAME

pthread_key_delete — thread-specific data key deletion

SYNOPSIS

#include <pthread.h>

int pthread_key_delete(pthread_key_t key);

DESCRIPTION

The *pthread_key_delete()* function shall delete a thread-specific data key previously returned by *pthread_key_create()*. The thread-specific data values associated with *key* need not be NULL at the time *pthread_key_delete()* is called. It is the responsibility of the application to free any application storage or perform any cleanup actions for data structures related to the deleted key or associated thread-specific data in any threads; this cleanup can be done either before or after *pthread_key_delete()* is called. Any attempt to use *key* following the call to *pthread_key_delete()* results in undefined behavior.

The *pthread_key_delete()* function shall be callable from within destructor functions. No destructor functions shall be invoked by *pthread_key_delete()*. Any destructor function that may have been associated with *key* shall no longer be called upon thread exit.

RETURN VALUE

If successful, the *pthread_key_delete()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_key_delete()* function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

A thread-specific data key deletion function has been included in order to allow the resources associated with an unused thread-specific data key to be freed. Unused thread-specific data keys can arise, among other scenarios, when a dynamically loaded module that allocated a key is unloaded.

Conforming applications are responsible for performing any cleanup actions needed for data structures associated with the key to be deleted, including data referenced by thread-specific data values. No such cleanup is done by *pthread_key_delete()*. In particular, destructor functions are not called. There are several reasons for this division of responsibility:

1. The associated destructor functions used to free thread-specific data at thread exit time are only guaranteed to work correctly when called in the thread that allocated the thread-specific data. (Destructors themselves may utilize thread-specific data.) Thus, they cannot be used to free thread-specific data in other threads at key deletion time. Attempting to have them called by other threads at key deletion time would require other threads to be asynchronously interrupted. But since interrupted threads could be in an arbitrary state, including holding locks necessary for the destructor to run, this approach would fail. In general, there is no safe mechanism whereby an implementation could free thread-specific data at key deletion time.
2. Even if there were a means of safely freeing thread-specific data associated with keys to be deleted, doing so would require that implementations be able to enumerate the threads with non-NULL data and potentially keep them from creating more thread-

pthread_key_delete()

System Interfaces

specific data while the key deletion is occurring. This special case could cause extra synchronization in the normal case, which would otherwise be unnecessary.

For an application to know that it is safe to delete a key, it has to know that all the threads that might potentially ever use the key do not attempt to use it again. For example, it could know this if all the client threads have called a cleanup procedure declaring that they are through with the module that is being shut down, perhaps by setting a reference count to zero.

If an implementation detects that the value specified by the *key* argument to *pthread_key_delete()* does not refer to a key value obtained from *pthread_key_create()* or refers to a key that has been deleted with *pthread_key_delete()*, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_key_create()

XBD <pthread.h>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_key_delete()* function is marked as part of the Threads option.

Issue 7

The *pthread_key_delete()* function is moved from the Threads option to the Base.

The [EINVAL] error for a key value not obtained from *pthread_key_create()* or a key deleted with *pthread_key_delete()* is removed; this condition results in undefined behavior.

52262 NAME

52263 pthread_kill — send a signal to a thread

52264 SYNOPSIS

```
52265 CX #include <signal.h>
52266 int pthread_kill(pthread_t thread, int sig);
```

52267 DESCRIPTION

52268 The *pthread_kill()* function shall request that a signal be delivered to the specified thread.

52269 As in *kill()*, if *sig* is zero, error checking shall be performed but no signal shall actually be sent.

52270 RETURN VALUE

52271 Upon successful completion, the function shall return a value of zero. Otherwise, the function
52272 shall return an error number. If the *pthread_kill()* function fails, no signal shall be sent.

52273 ERRORS

52274 The *pthread_kill()* function shall fail if:

52275 [EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.

52276 The *pthread_kill()* function shall not return an error code of [EINTR].

52277 EXAMPLES

52278 None.

52279 APPLICATION USAGE

52280 The *pthread_kill()* function provides a mechanism for asynchronously directing a signal at a
52281 thread in the calling process. This could be used, for example, by one thread to affect broadcast
52282 delivery of a signal to a set of threads.

52283 Note that *pthread_kill()* only causes the signal to be handled in the context of the given thread;
52284 the signal action (termination or stopping) affects the process as a whole.

52285 RATIONALE

52286 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
52287 that the function should fail and report an [ESRCH] error.

52288 FUTURE DIRECTIONS

52289 None.

52290 SEE ALSO

52291 *kill()*, *pthread_self()*, *raise()*

52292 XBD <signal.h>

52293 CHANGE HISTORY

52294 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

52295 Issue 6

52296 The *pthread_kill()* function is marked as part of the Threads option.

52297 The APPLICATION USAGE section is added.

52298 Issue 7

52299 The *pthread_kill()* function is moved from the Threads option to the Base.

52300 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

pthread_mutex_consistent()

System Interfaces

52301 NAME

52302 pthread_mutex_consistent — mark state protected by robust mutex as consistent

52303 SYNOPSIS

52304 #include <pthread.h>

52305 int pthread_mutex_consistent(pthread_mutex_t *mutex);

52306 DESCRIPTION52307 If *mutex* is a robust mutex in an inconsistent state, the *pthread_mutex_consistent()* function can be
52308 used to mark the state protected by the mutex referenced by *mutex* as consistent again.52309 If an owner of a robust mutex terminates while holding the mutex, the mutex becomes
52310 inconsistent and the next thread that acquires the mutex lock shall be notified of the state by the
52311 return value [EOWNERDEAD]. In this case, the mutex does not become normally usable again
52312 until the state is marked consistent.52313 If the thread which acquired the mutex lock with the return value [EOWNERDEAD] terminates
52314 before calling either *pthread_mutex_consistent()* or *pthread_mutex_unlock()*, the next thread that
52315 acquires the mutex lock shall be notified about the state of the mutex by the return value
52316 [EOWNERDEAD].52317 The behavior is undefined if the value specified by the *mutex* argument to
52318 *pthread_mutex_consistent()* does not refer to an initialized mutex.**52319 RETURN VALUE**52320 Upon successful completion, the *pthread_mutex_consistent()* function shall return zero.
52321 Otherwise, an error value shall be returned to indicate the error.**52322 ERRORS**52323 The *pthread_mutex_consistent()* function shall fail if:52324 [EINVAL] The mutex object referenced by *mutex* is not robust or does not protect an
52325 inconsistent state.

52326 These functions shall not return an error code of [EINTR].

52327 EXAMPLES

52328 None.

52329 APPLICATION USAGE52330 The *pthread_mutex_consistent()* function is only responsible for notifying the implementation that
52331 the state protected by the mutex has been recovered and that normal operations with the mutex
52332 can be resumed. It is the responsibility of the application to recover the state so it can be reused.
52333 If the application is not able to perform the recovery, it can notify the implementation that the
52334 situation is unrecoverable by a call to *pthread_mutex_unlock()* without a prior call to
52335 *pthread_mutex_consistent()*, in which case subsequent threads that attempt to lock the mutex will
52336 fail to acquire the lock and be returned [ENOTRECOVERABLE].**52337 RATIONALE**52338 If an implementation detects that the value specified by the *mutex* argument to
52339 *pthread_mutex_consistent()* does not refer to an initialized mutex, it is recommended that the
52340 function should fail and report an [EINVAL] error.**52341 FUTURE DIRECTIONS**

52342 None.

52343 **SEE ALSO**52344 [pthread_mutex_lock\(\)](#), [pthread_mutexattr_getrobust\(\)](#)52345 XBD [<pthread.h>](#)52346 **CHANGE HISTORY**

52347 First released in Issue 7.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_mutex_destroy()

System Interfaces

52348 **NAME**

52349 pthread_mutex_destroy, pthread_mutex_init — destroy and initialize a mutex

52350 **SYNOPSIS**

```

52351 #include <pthread.h>

52352 int pthread_mutex_destroy(pthread_mutex_t *mutex);
52353 int pthread_mutex_init(pthread_mutex_t *restrict mutex,
52354     const pthread_mutexattr_t *restrict attr);
52355 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

```

52356 **DESCRIPTION**

52357 The *pthread_mutex_destroy()* function shall destroy the mutex object referenced by *mutex*; the
 52358 mutex object becomes, in effect, uninitialized. An implementation may cause
 52359 *pthread_mutex_destroy()* to set the object referenced by *mutex* to an invalid value.

52360 A destroyed mutex object can be reinitialized using *pthread_mutex_init()*; the results of otherwise
 52361 referencing the object after it has been destroyed are undefined.

52362 It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked
 52363 mutex or a mutex that is referenced (for example, while being used in a *pthread_cond_timedwait()*
 52364 or *pthread_cond_wait()*) by another thread results in undefined behavior.

52365 The *pthread_mutex_init()* function shall initialize the mutex referenced by *mutex* with attributes
 52366 specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the
 52367 same as passing the address of a default mutex attributes object. Upon successful initialization,
 52368 the state of the mutex becomes initialized and unlocked.

52369 Only *mutex* itself may be used for performing synchronization. The result of referring to copies
 52370 of *mutex* in calls to *pthread_mutex_lock()*, *pthread_mutex_trylock()*, *pthread_mutex_unlock()*, and
 52371 *pthread_mutex_destroy()* is undefined.

52372 Attempting to initialize an already initialized mutex results in undefined behavior.

52373 In cases where default mutex attributes are appropriate, the macro
 52374 PTHREAD_MUTEX_INITIALIZER can be used to initialize mutexes that are statically allocated.
 52375 The effect shall be equivalent to dynamic initialization by a call to *pthread_mutex_init()* with
 52376 parameter *attr* specified as NULL, except that no error checks are performed.

52377 The behavior is undefined if the value specified by the *mutex* argument to
 52378 *pthread_mutex_destroy()* does not refer to an initialized mutex.

52379 The behavior is undefined if the value specified by the *attr* argument to *pthread_mutex_init()*
 52380 does not refer to an initialized mutex attributes object.

52381 **RETURN VALUE**

52382 If successful, the *pthread_mutex_destroy()* and *pthread_mutex_init()* functions shall return zero;
 52383 otherwise, an error number shall be returned to indicate the error.

52384 **ERRORS**

52385 The *pthread_mutex_init()* function shall fail if:

52386	[EAGAIN]	The system lacked the necessary resources (other than memory) to initialize another mutex.
52387		
52388	[ENOMEM]	Insufficient memory exists to initialize the mutex.
52389	[EPERM]	The caller does not have the privilege to perform the operation.

52390 The *pthread_mutex_init()* function may fail if:

52391 [EINVAL] The attributes object referenced by *attr* has the robust mutex attribute set
52392 without the process-shared attribute being set.

52393 These functions shall not return an error code of [EINTR].

52394 EXAMPLES

52395 None.

52396 APPLICATION USAGE

52397 None.

52398 RATIONALE

52399 If an implementation detects that the value specified by the *mutex* argument to
52400 *pthread_mutex_destroy()* does not refer to an initialized mutex, it is recommended that the
52401 function should fail and report an [EINVAL] error.

52402 If an implementation detects that the value specified by the *mutex* argument to
52403 *pthread_mutex_destroy()* or *pthread_mutex_init()* refers to a locked mutex or a mutex that is
52404 referenced (for example, while being used in a *pthread_cond_timedwait()* or *pthread_cond_wait()*)
52405 by another thread, or detects that the value specified by the *mutex* argument to
52406 *pthread_mutex_init()* refers to an already initialized mutex, it is recommended that the function
52407 should fail and report an [EBUSY] error.

52408 If an implementation detects that the value specified by the *attr* argument to
52409 *pthread_mutex_init()* does not refer to an initialized mutex attributes object, it is recommended
52410 that the function should fail and report an [EINVAL] error.

52411 Alternate Implementations Possible

52412 This volume of POSIX.1-2008 supports several alternative implementations of mutexes. An
52413 implementation may store the lock directly in the object of type **pthread_mutex_t**. Alternatively,
52414 an implementation may store the lock in the heap and merely store a pointer, handle, or unique
52415 ID in the mutex object. Either implementation has advantages or may be required on certain
52416 hardware configurations. So that portable code can be written that is invariant to this choice, this
52417 volume of POSIX.1-2008 does not define assignment or equality for this type, and it uses the
52418 term “initialize” to reinforce the (more restrictive) notion that the lock may actually reside in the
52419 mutex object itself.

52420 Note that this precludes an over-specification of the type of the mutex or condition variable and
52421 motivates the opaqueness of the type.

52422 An implementation is permitted, but not required, to have *pthread_mutex_destroy()* store an
52423 illegal value into the mutex. This may help detect erroneous programs that try to lock (or
52424 otherwise reference) a mutex that has already been destroyed.

52425 Tradeoff Between Error Checks and Performance Supported

52426 Many error conditions that can occur are not required to be detected by the implementation in
52427 order to let implementations trade off performance *versus* degree of error checking according to
52428 the needs of their specific applications and execution environment. As a general rule, conditions
52429 caused by the system (such as insufficient memory) are required to be detected, but conditions
52430 caused by an erroneously coded application (such as failing to provide adequate
52431 synchronization to prevent a mutex from being deleted while in use) are specified to result in
52432 undefined behavior.

52433 A wide range of implementations is thus made possible. For example, an implementation

pthread_mutex_destroy()*System Interfaces*

intended for application debugging may implement all of the error checks, but an implementation running a single, provably correct application under very tight performance constraints in an embedded computer might implement minimal checks. An implementation might even be provided in two versions, similar to the options that compilers provide: a full-checking, but slower version; and a limited-checking, but faster version. To forbid this optionality would be a disservice to users.

By carefully limiting the use of “undefined behavior” only to things that an erroneous (badly coded) application might do, and by defining that resource-not-available errors are mandatory, this volume of POSIX.1-2008 ensures that a fully-conforming application is portable across the full range of implementations, while not forcing all implementations to add overhead to check for numerous things that a correct program never does. When the behavior is undefined, no error number is specified to be returned on implementations that do detect the condition. This is because undefined behavior means *anything* can happen, which includes returning with any value (which might happen to be a valid, but different, error number). However, since the error number might be useful to application developers when diagnosing problems during application development, a recommendation is made in rationale that implementors should return a particular error number if their implementation does detect the condition.

Why No Limits are Defined

Defining symbols for the maximum number of mutexes and condition variables was considered but rejected because the number of these objects may change dynamically. Furthermore, many implementations place these objects into application memory; thus, there is no explicit maximum.

Static Initializers for Mutexes and Condition Variables

Providing for static initialization of statically allocated synchronization objects allows modules with private static synchronization variables to avoid runtime initialization tests and overhead. Furthermore, it simplifies the coding of self-initializing modules. Such modules are common in C libraries, where for various reasons the design calls for self-initialization instead of requiring an explicit module initialization function to be called. An example use of static initialization follows.

Without static initialization, a self-initializing routine *foo()* might look as follows:

```
static pthread_once_t foo_once = PTHREAD_ONCE_INIT;
static pthread_mutex_t foo_mutex;

void foo_init()
{
    pthread_mutex_init(&foo_mutex, NULL);
}

void foo()
{
    pthread_once(&foo_once, foo_init);
    pthread_mutex_lock(&foo_mutex);
    /* Do work. */
    pthread_mutex_unlock(&foo_mutex);
}
```

With static initialization, the same routine could be coded as follows:

```
static pthread_mutex_t foo_mutex = PTHREAD_MUTEX_INITIALIZER;
```

```

52479 void foo()
52480 {
52481     pthread_mutex_lock(&foo_mutex);
52482     /* Do work. */
52483     pthread_mutex_unlock(&foo_mutex);
52484 }

```

Note that the static initialization both eliminates the need for the initialization test inside *pthread_once()* and the fetch of *&foo_mutex* to learn the address to be passed to *pthread_mutex_lock()* or *pthread_mutex_unlock()*.

Thus, the C code written to initialize static objects is simpler on all systems and is also faster on a large class of systems; those where the (entire) synchronization object can be stored in application memory.

Yet the locking performance question is likely to be raised for machines that require mutexes to be allocated out of special memory. Such machines actually have to have mutexes and possibly condition variables contain pointers to the actual hardware locks. For static initialization to work on such machines, *pthread_mutex_lock()* also has to test whether or not the pointer to the actual lock has been allocated. If it has not, *pthread_mutex_lock()* has to initialize it before use. The reservation of such resources can be made when the program is loaded, and hence return codes have not been added to mutex locking and condition variable waiting to indicate failure to complete initialization.

This runtime test in *pthread_mutex_lock()* would at first seem to be extra work; an extra test is required to see whether the pointer has been initialized. On most machines this would actually be implemented as a fetch of the pointer, testing the pointer against zero, and then using the pointer if it has already been initialized. While the test might seem to add extra work, the extra effort of testing a register is usually negligible since no extra memory references are actually done. As more and more machines provide caches, the real expenses are memory references, not instructions executed.

Alternatively, depending on the machine architecture, there are often ways to eliminate *all* overhead in the most important case: on the lock operations that occur *after* the lock has been initialized. This can be done by shifting more overhead to the less frequent operation: initialization. Since out-of-line mutex allocation also means that an address has to be dereferenced to find the actual lock, one technique that is widely applicable is to have static initialization store a bogus value for that address; in particular, an address that causes a machine fault to occur. When such a fault occurs upon the first attempt to lock such a mutex, validity checks can be done, and then the correct address for the actual lock can be filled in. Subsequent lock operations incur no extra overhead since they do not “fault”. This is merely one technique that can be used to support static initialization, while not adversely affecting the performance of lock acquisition. No doubt there are other techniques that are highly machine-dependent.

The locking overhead for machines doing out-of-line mutex allocation is thus similar for modules being implicitly initialized, where it is improved for those doing mutex allocation entirely inline. The inline case is thus made much faster, and the out-of-line case is not significantly worse.

Besides the issue of locking performance for such machines, a concern is raised that it is possible that threads would serialize contending for initialization locks when attempting to finish initializing statically allocated mutexes. (Such finishing would typically involve taking an internal lock, allocating a structure, storing a pointer to the structure in the mutex, and releasing the internal lock.) First, many implementations would reduce such serialization by hashing on the mutex address. Second, such serialization can only occur a bounded number of times. In

pthread_mutex_destroy()

System Interfaces

particular, it can happen at most as many times as there are statically allocated synchronization objects. Dynamically allocated objects would still be initialized via *pthread_mutex_init()* or *pthread_cond_init()*.

Finally, if none of the above optimization techniques for out-of-line allocation yields sufficient performance for an application on some implementation, the application can avoid static initialization altogether by explicitly initializing all synchronization objects with the corresponding *pthread_*_init()* functions, which are supported by all implementations. An implementation can also document the tradeoffs and advise which initialization technique is more efficient for that particular implementation.

Destroying Mutexes

A mutex can be destroyed immediately after it is unlocked. For example, consider the following code:

```
struct obj {
pthread_mutex_t om;
    int refcnt;
    ...
};

obj_done(struct obj *op)
{
    pthread_mutex_lock(&op->om);
    if (--op->refcnt == 0) {
        pthread_mutex_unlock(&op->om);
        (A) pthread_mutex_destroy(&op->om);
        (B) free(op);
    } else
        (C) pthread_mutex_unlock(&op->om);
}
```

In this case *obj* is reference counted and *obj_done()* is called whenever a reference to the object is dropped. Implementations are required to allow an object to be destroyed and freed and potentially unmapped (for example, lines A and B) immediately after the object is unlocked (line C).

Robust Mutexes

Implementations are required to provide robust mutexes for mutexes with the process-shared attribute set to *PTHREAD_PROCESS_SHARED*. Implementations are allowed, but not required, to provide robust mutexes when the process-shared attribute is set to *PTHREAD_PROCESS_PRIVATE*.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_getprioceiling(), *pthread_mutexattr_getrobust()*, *pthread_mutex_lock()*, *pthread_mutex_timedlock()*, *pthread_mutexattr_getpshared()*

XBD [<pthread.h>](#)

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_mutex_destroy()* and *pthread_mutex_init()* functions are marked as part of the Threads option.

The *pthread_mutex_timedlock()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

The **restrict** keyword is added to the *pthread_mutex_init()* prototype for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_mutex_destroy()* and *pthread_mutex_init()* functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized mutex or an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for a locked mutex, a mutex that is referenced, or an already initialized mutex is removed; this condition results in undefined behavior.

pthread_mutex_getprioceiling()

System Interfaces

52587 **NAME**

52588 pthread_mutex_getprioceiling, pthread_mutex_setprioceiling — get and set the priority ceiling
 52589 of a mutex (**REALTIME THREADS**)

52590 **SYNOPSIS**

```
52591 RPP|TPP #include <pthread.h>
52592
52592 int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
52593 int *restrict prioceiling);
52594
52594 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
52595 int prioceiling, int *restrict old_ceiling);
```

52596 **DESCRIPTION**

52597 The *pthread_mutex_getprioceiling()* function shall return the current priority ceiling of the mutex.

52598 The *pthread_mutex_setprioceiling()* function shall attempt to lock the mutex as if by a call to
 52599 *pthread_mutex_lock()*, except that the process of locking the mutex need not adhere to the priority
 52600 protect protocol. On acquiring the mutex it shall change the mutex's priority ceiling and then
 52601 release the mutex as if by a call to *pthread_mutex_unlock()*. When the change is successful, the
 52602 previous value of the priority ceiling shall be returned in *old_ceiling*.

52603 If the *pthread_mutex_setprioceiling()* function fails, the mutex priority ceiling shall not be
 52604 changed.

52605 **RETURN VALUE**

52606 If successful, the *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions shall
 52607 return zero; otherwise, an error number shall be returned to indicate the error.

52608 **ERRORS**

52609 These functions shall fail if:

52610 [EINVAL] The protocol attribute of *mutex* is PTHREAD_PRIO_NONE.

52611 [EPERM] The implementation requires appropriate privileges to perform the operation
 52612 and the caller does not have appropriate privileges.

52613 The *pthread_mutex_setprioceiling()* function shall fail if:

52614 [EAGAIN] The mutex could not be acquired because the maximum number of recursive
 52615 locks for *mutex* has been exceeded.

52616 [EDEADLK] The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current
 52617 thread already owns the mutex.

52618 [EINVAL] The mutex was created with the protocol attribute having the value
 52619 PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than
 52620 the mutex's current priority ceiling, and the implementation adheres to the
 52621 priority protect protocol in the process of locking the mutex.

52622 [ENOTRECOVERABLE]

52623 The mutex is a robust mutex and the state protected by the mutex is not
 52624 recoverable.

52625 [EOWNERDEAD]

52626 The mutex is a robust mutex and the process containing the previous owning
 52627 thread terminated while holding the mutex lock. The mutex lock shall be
 52628 acquired by the calling thread and it is up to the new owner to make the state
 52629 consistent (see *pthread_mutex_lock()*).

- 52630 The *pthread_mutex_setprioceiling()* function may fail if:
- 52631 [EDEADLK] A deadlock condition was detected.
- 52632 [EINVAL] The priority requested by *prioceiling* is out of range.
- 52633 [EOWNERDEAD]
- 52634 The mutex is a robust mutex and the previous owning thread terminated
- 52635 while holding the mutex lock. The mutex lock shall be acquired by the calling
- 52636 thread and it is up to the new owner to make the state consistent (see
- 52637 *pthread_mutex_lock()*).
- 52638 These functions shall not return an error code of [EINTR].
- 52639 **EXAMPLES**
- 52640 None.
- 52641 **APPLICATION USAGE**
- 52642 None.
- 52643 **RATIONALE**
- 52644 None.
- 52645 **FUTURE DIRECTIONS**
- 52646 None.
- 52647 **SEE ALSO**
- 52648 *pthread_mutex_destroy()*, *pthread_mutex_lock()*, *pthread_mutex_timedlock()*
- 52649 XBD **<pthread.h>**
- 52650 **CHANGE HISTORY**
- 52651 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
- 52652 Marked as part of the Realtime Threads Feature Group.
- 52653 **Issue 6**
- 52654 The *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions are marked as
- 52655 part of the Threads and Thread Priority Protection options.
- 52656 The [ENOSYS] error conditions have been removed.
- 52657 The *pthread_mutex_timedlock()* function is added to the SEE ALSO section for alignment with
- 52658 IEEE Std 1003.1d-1999.
- 52659 The **restrict** keyword is added to the *pthread_mutex_getprioceiling()* and
- 52660 *pthread_mutex_setprioceiling()* prototypes for alignment with the ISO/IEC 9899:1999 standard.
- 52661 **Issue 7**
- 52662 SD5-XSH-ERN-39 is applied.
- 52663 Austin Group Interpretation 1003.1-2001 #052 is applied, adding [EDEADLK] as a “may fail”
- 52664 error.
- 52665 SD5-XSH-ERN-158 is applied, updating the ERRORS section to include a “shall fail” error case
- 52666 for when the protocol attribute of *mutex* is PTHREAD_PRIO_NONE.
- 52667 The *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions are moved from
- 52668 the Threads option to require support of either the Robust Mutex Priority Protection option or
- 52669 the Non-Robust Mutex Priority Protection option.

pthread_mutex_getprioceiling()

System Interfaces

52670
52671

The DESCRIPTION and ERRORS sections are updated to account properly for all of the various mutex types.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread_mutex_init()**52672 **NAME**

52673 pthread_mutex_init — destroy and initialize a mutex

52674 **SYNOPSIS**

52675 #include <pthread.h>

52676 int pthread_mutex_init(pthread_mutex_t *restrict mutex,

52677 const pthread_mutexattr_t *restrict attr);

52678 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

52679 **DESCRIPTION**52680 Refer to *pthread_mutex_destroy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_mutex_lock()

System Interfaces

52681 **NAME**

52682 pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a
 52683 mutex

52684 **SYNOPSIS**

```
52685 #include <pthread.h>

52686 int pthread_mutex_lock(pthread_mutex_t *mutex);
52687 int pthread_mutex_trylock(pthread_mutex_t *mutex);
52688 int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

52689 **DESCRIPTION**

52690 The mutex object referenced by *mutex* shall be locked by calling *pthread_mutex_lock()*. If the
 52691 mutex is already locked, the calling thread shall block until the mutex becomes available. This
 52692 operation shall return with the mutex object referenced by *mutex* in the locked state with the
 52693 calling thread as its owner.

52694 If the mutex type is PTHREAD_MUTEX_NORMAL, deadlock detection shall not be provided.
 52695 Attempting to relock the mutex causes deadlock. If a thread attempts to unlock a mutex that it
 52696 has not locked or a mutex which is unlocked, undefined behavior results.

52697 If the mutex type is PTHREAD_MUTEX_ERRORCHECK, then error checking shall be provided.
 52698 If a thread attempts to relock a mutex that it has already locked, an error shall be returned. If a
 52699 thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error
 52700 shall be returned.

52701 If the mutex type is PTHREAD_MUTEX_RECURSIVE, then the mutex shall maintain the
 52702 concept of a lock count. When a thread successfully acquires a mutex for the first time, the lock
 52703 count shall be set to one. Every time a thread relocks this mutex, the lock count shall be
 52704 incremented by one. Each time the thread unlocks the mutex, the lock count shall be
 52705 decremented by one. When the lock count reaches zero, the mutex shall become available for
 52706 other threads to acquire. If a thread attempts to unlock a mutex that it has not locked or a mutex
 52707 which is unlocked, an error shall be returned.

52708 If the mutex type is PTHREAD_MUTEX_DEFAULT, attempting to recursively lock the mutex
 52709 results in undefined behavior. Attempting to unlock the mutex if it was not locked by the calling
 52710 thread results in undefined behavior. Attempting to unlock the mutex if it is not locked results in
 52711 undefined behavior.

52712 The *pthread_mutex_trylock()* function shall be equivalent to *pthread_mutex_lock()*, except that if
 52713 the mutex object referenced by *mutex* is currently locked (by any thread, including the current
 52714 thread), the call shall return immediately. If the mutex type is PTHREAD_MUTEX_RECURSIVE
 52715 and the mutex is currently owned by the calling thread, the mutex lock count shall be
 52716 incremented by one and the *pthread_mutex_trylock()* function shall immediately return success.

52717 The *pthread_mutex_unlock()* function shall release the mutex object referenced by *mutex*. The
 52718 manner in which a mutex is released is dependent upon the mutex's type attribute. If there are
 52719 threads blocked on the mutex object referenced by *mutex* when *pthread_mutex_unlock()* is called,
 52720 resulting in the mutex becoming available, the scheduling policy shall determine which thread
 52721 shall acquire the mutex.

52722 (In the case of PTHREAD_MUTEX_RECURSIVE mutexes, the mutex shall become available
 52723 when the count reaches zero and the calling thread no longer has any locks on this mutex.)

52724 If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the
 52725 thread shall resume waiting for the mutex as if it was not interrupted.

52726 If *mutex* is a robust mutex and the process containing the owning thread terminated while

holding the mutex lock, a call to *pthread_mutex_lock()* shall return the error value [EOWNERDEAD]. If *mutex* is a robust mutex and the owning thread terminated while holding the mutex lock, a call to *pthread_mutex_lock()* may return the error value [EOWNERDEAD] even if the process in which the owning thread resides has not terminated. In these cases, the mutex is locked by the thread but the state it protects is marked as inconsistent. The application should ensure that the state is made consistent for reuse and when that is complete call *pthread_mutex_consistent()*. If the application is unable to recover the state, it should unlock the mutex without a prior call to *pthread_mutex_consistent()*, after which the mutex is marked permanently unusable.

If *mutex* does not refer to an initialized mutex object, the behavior of *pthread_mutex_lock()*, *pthread_mutex_trylock()*, and *pthread_mutex_unlock()* is undefined.

RETURN VALUE

If successful, the *pthread_mutex_lock()* and *pthread_mutex_unlock()* functions shall return zero; otherwise, an error number shall be returned to indicate the error.

The *pthread_mutex_trylock()* function shall return zero if a lock on the mutex object referenced by *mutex* is acquired. Otherwise, an error number is returned to indicate the error.

ERRORS

The *pthread_mutex_lock()* and *pthread_mutex_trylock()* functions shall fail if:

[EAGAIN] The mutex could not be acquired because the maximum number of recursive locks for *mutex* has been exceeded.

RPP | TPP [EINVAL] The *mutex* was created with the protocol attribute having the value PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than the mutex's current priority ceiling.

[ENOTRECOVERABLE]

The state protected by the mutex is not recoverable.

[EOWNERDEAD]

The mutex is a robust mutex and the process containing the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

The *pthread_mutex_lock()* function shall fail if:

[EDEADLK] The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current thread already owns the mutex.

The *pthread_mutex_trylock()* function shall fail if:

[EBUSY] The *mutex* could not be acquired because it was already locked.

The *pthread_mutex_unlock()* function shall fail if:

[EPERM] The mutex type is PTHREAD_MUTEX_ERRORCHECK or the mutex is a robust mutex, and the current thread does not own the mutex.

The *pthread_mutex_lock()* and *pthread_mutex_trylock()* functions may fail if:

[EOWNERDEAD]

The mutex is a robust mutex and the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

pthread_mutex_lock()*System Interfaces*

52770 The *pthread_mutex_lock()* function may fail if:

52771 [EDEADLK] A deadlock condition was detected.

52772 These functions shall not return an error code of [EINTR].

EXAMPLES

52773 None.

APPLICATION USAGE

52776 Applications that have assumed that non-zero return values are errors will need updating for use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If an application is supposed to work with normal and robust mutexes it should check all return values for error conditions and if necessary take appropriate action.

RATIONALE

52783 Mutex objects are intended to serve as a low-level primitive from which other thread synchronization functions can be built. As such, the implementation of mutexes should be as efficient as possible, and this has ramifications on the features available at the interface.

52786 The mutex functions and the particular default settings of the mutex attributes have been motivated by the desire to not preclude fast, inlined implementations of mutex locking and unlocking.

52789 Since most attributes only need to be checked when a thread is going to be blocked, the use of attributes does not slow the (common) mutex-locking case.

52791 Likewise, while being able to extract the thread ID of the owner of a mutex might be desirable, it would require storing the current thread ID when each mutex is locked, and this could incur unacceptable levels of overhead. Similar arguments apply to a *mutex_tryunlock* operation.

52794 For further rationale on the extended mutex types, see XRAT [Threads Extensions](#) (on page 3573).

52795 If an implementation detects that the value specified by the *mutex* argument does not refer to an initialized mutex object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

52799 None.

SEE ALSO

52801 [pthread_mutex_consistent\(\)](#), [pthread_mutex_destroy\(\)](#), [pthread_mutex_timedlock\(\)](#),
52802 [pthread_mutexattr_getrobust\(\)](#)

52803 XBD Section 4.11 (on page 110), [<pthread.h>](#)

CHANGE HISTORY

52805 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

52807 The *pthread_mutex_lock()*, *pthread_mutex_trylock()*, and *pthread_mutex_unlock()* functions are marked as part of the Threads option.

- 52809 The following new requirements on POSIX implementations derive from alignment with the
52810 Single UNIX Specification:
- 52811 • The behavior when attempting to relock a mutex is defined.
- 52812 The *pthread_mutex_timedlock()* function is added to the SEE ALSO section for alignment with
52813 IEEE Std 1003.1d-1999.
- 52814 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/98 is applied, updating the ERRORS
52815 section so that the [EDEADLK] error includes detection of a deadlock condition. The
52816 RATIONALE section is also reworded to take into account non-XSI-conformant systems.
- 52817 **Issue 7**
- 52818 SD5-XSH-ERN-43 is applied, marking the “shall fail” case of the [EINVAL] error as dependent
52819 on the Thread Priority Protection option.
- 52820 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.
- 52821 The *pthread_mutex_lock()*, *pthread_mutex_trylock()*, and *pthread_mutex_unlock()* functions are
52822 moved from the Threads option to the Base.
- 52823 The following extended mutex types are moved from the XSI option to the Base:
- 52824 PTHREAD_MUTEX_NORMAL
 - 52825 PTHREAD_MUTEX_ERRORCHECK
 - 52826 PTHREAD_MUTEX_RECURSIVE
 - 52827 PTHREAD_MUTEX_DEFAULT
- 52828 The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized
52829 mutex.
- 52830 The ERRORS section is updated to account properly for all of the various mutex types.

pthread_mutex_setprioceiling()*System Interfaces*52831 **NAME**

52832 pthread_mutex_setprioceiling — change the priority ceiling of a mutex (**REALTIME**
52833 **THREADS**)

52834 **SYNOPSIS**

```
52835 RPP|TPP #include <pthread.h>  
52836 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,  
52837 int prioceiling, int *restrict old_ceiling);
```

52838 **DESCRIPTION**

52839 Refer to *pthread_mutex_getprioceiling()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

pthread_mutex_timedlock()**NAME**

pthread_mutex_timedlock — lock a mutex

SYNOPSIS

```
#include <pthread.h>
#include <time.h>

int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abstime);
```

DESCRIPTION

The *pthread_mutex_timedlock()* function shall lock the mutex object referenced by *mutex*. If the mutex is already locked, the calling thread shall block until the mutex becomes available as in the *pthread_mutex_lock()* function. If the mutex cannot be locked without waiting for another thread to unlock the mutex, this wait shall be terminated when the specified timeout expires.

The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the clock on which it is based. The *timespec* data type is defined in the *<time.h>* header.

Under no circumstance shall the function fail with a timeout if the mutex can be locked immediately. The validity of the *abstime* parameter need not be checked if the mutex can be locked immediately.

RPI|TPI

As a consequence of the priority inheritance rules (for mutexes initialized with the PRIO_INHERIT protocol), if a timed mutex wait is terminated because its timeout expires, the priority of the owner of the mutex shall be adjusted as necessary to reflect the fact that this thread is no longer among the threads waiting for the mutex.

If *mutex* is a robust mutex and the process containing the owning thread terminated while holding the mutex lock, a call to *pthread_mutex_timedlock()* shall return the error value [EOWNERDEAD]. If *mutex* is a robust mutex and the owning thread terminated while holding the mutex lock, a call to *pthread_mutex_timedlock()* may return the error value [EOWNERDEAD] even if the process in which the owning thread resides has not terminated. In these cases, the mutex is locked by the thread but the state it protects is marked as inconsistent. The application should ensure that the state is made consistent for reuse and when that is complete call *pthread_mutex_consistent()*. If the application is unable to recover the state, it should unlock the mutex without a prior call to *pthread_mutex_consistent()*, after which the mutex is marked permanently unusable.

If *mutex* does not refer to an initialized mutex object, the behavior is undefined.

RETURN VALUE

If successful, the *pthread_mutex_timedlock()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutex_timedlock()* function shall fail if:

[EAGAIN] The mutex could not be acquired because the maximum number of recursive locks for *mutex* has been exceeded.

pthread_mutex_timedlock()

System Interfaces

52884	[EDEADLK]	The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current thread already owns the mutex.
52885		
52886	[EINVAL]	The mutex was created with the protocol attribute having the value PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than the mutex' current priority ceiling.
52887		
52888		
52889	[EINVAL]	The process or thread would have blocked, and the <i>abstime</i> parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.
52890		
52891		
52892	[ENOTRECOVERABLE]	
52893		The state protected by the mutex is not recoverable.
52894	[EOWNERDEAD]	
52895		The mutex is a robust mutex and the process containing the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.
52896		
52897		
52898		
52899	[ETIMEDOUT]	The mutex could not be locked before the specified timeout expired.
52900		The <i>pthread_mutex_timedlock()</i> function may fail if:
52901	[EDEADLK]	A deadlock condition was detected.
52902	[EOWNERDEAD]	
52903		The mutex is a robust mutex and the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.
52904		
52905		
52906		This function shall not return an error code of [EINTR].

EXAMPLES

52907
52908 None.

APPLICATION USAGE

52909 Applications that have assumed that non-zero return values are errors will need updating for
52910 use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting
52911 a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error
52912 returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If
52913 an application is supposed to work with normal and robust mutexes, it should check all return
52914 values for error conditions and if necessary take appropriate action.
52915

RATIONALE

52916 Refer to *pthread_mutex_lock()*.
52917

FUTURE DIRECTIONS

52918
52919 None.

SEE ALSO

52920 *pthread_mutex_destroy()*, *pthread_mutex_lock()*, *time()*
52921
52922 XBD Section 4.11 (on page 110), *<pthread.h>*, *<time.h>*

CHANGE HISTORY

52923 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
52924

52925 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/99 is applied, marking the last paragraph
52926 in the DESCRIPTION as part of the Thread Priority Inheritance option.

- 52927 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/100 is applied, updating the ERRORS
52928 section so that the [EDEADLK] error includes detection of a deadlock condition.
- 52929 **Issue 7**
52930 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.
- 52931 The *pthread_mutex_timedlock()* function is moved from the Timeouts option to the Base.
- 52932 Functionality relating to the Timers option is moved to the Base.
- 52933 The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized
52934 mutex.
- 52935 The ERRORS section is updated to account properly for all of the various mutex types.

pthread_mutex_trylock()*System Interfaces*52936 **NAME**

52937 pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a mutex

52938 **SYNOPSIS**

52939 #include <pthread.h>

52940 int pthread_mutex_trylock(pthread_mutex_t *mutex);

52941 int pthread_mutex_unlock(pthread_mutex_t *mutex);

52942 **DESCRIPTION**52943 Refer to *pthread_mutex_lock()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

`pthread_mutexattr_destroy`, `pthread_mutexattr_init` — destroy and initialize the mutex attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

DESCRIPTION

The `pthread_mutexattr_destroy()` function shall destroy a mutex attributes object; the object becomes, in effect, uninitialized. An implementation may cause `pthread_mutexattr_destroy()` to set the object referenced by `attr` to an invalid value.

A destroyed `attr` attributes object can be reinitialized using `pthread_mutexattr_init()`; the results of otherwise referencing the object after it has been destroyed are undefined.

The `pthread_mutexattr_init()` function shall initialize a mutex attributes object `attr` with the default value for all of the attributes defined by the implementation.

Results are undefined if `pthread_mutexattr_init()` is called specifying an already initialized `attr` attributes object.

After a mutex attributes object has been used to initialize one or more mutexes, any function affecting the attributes object (including destruction) shall not affect any previously initialized mutexes.

The behavior is undefined if the value specified by the `attr` argument to `pthread_mutexattr_destroy()` does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, `pthread_mutexattr_destroy()` and `pthread_mutexattr_init()` shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The `pthread_mutexattr_init()` function shall fail if:

[ENOMEM] Insufficient memory exists to initialize the mutex attributes object.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the `attr` argument to `pthread_mutexattr_destroy()` does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an [EINVAL] error.

See `pthread_attr_destroy()` for a general explanation of attributes. Attributes objects allow implementations to experiment with useful extensions and permit extension of this volume of POSIX.1-2008 without changing the existing functions. Thus, they provide for future extensibility of this volume of POSIX.1-2008 and reduce the temptation to standardize prematurely on semantics that are not yet widely implemented or understood.

Examples of possible additional mutex attributes that have been discussed are *spin_only*, *limited_spin*, *no_spin*, *recursive*, and *metered*. (To explain what the latter attributes might mean:

pthread_mutexattr_destroy()*System Interfaces*

recursive mutexes would allow for multiple re-locking by the current owner; metered mutexes would transparently keep records of queue length, wait time, and so on.) Since there is not yet wide agreement on the usefulness of these resulting from shared implementation and usage experience, they are not yet specified in this volume of POSIX.1-2008. Mutex attributes objects, however, make it possible to test out these concepts for possible standardization at a later time.

Mutex Attributes and Performance

Care has been taken to ensure that the default values of the mutex attributes have been defined such that mutexes initialized with the defaults have simple enough semantics so that the locking and unlocking can be done with the equivalent of a test-and-set instruction (plus possibly a few other basic instructions).

There is at least one implementation method that can be used to reduce the cost of testing at lock-time if a mutex has non-default attributes. One such method that an implementation can employ (and this can be made fully transparent to fully conforming POSIX applications) is to secretly pre-lock any mutexes that are initialized to non-default attributes. Any later attempt to lock such a mutex causes the implementation to branch to the “slow-path” as if the mutex were unavailable; then, on the slow path, the implementation can do the “real work” to lock a non-default mutex. The underlying unlock operation is more complicated since the implementation never really wants to release the pre-lock on this kind of mutex. This illustrates that, depending on the hardware, there may be certain optimizations that can be used so that whatever mutex attributes are considered “most frequently used” can be processed most efficiently.

Process Shared Memory and Synchronization

The existence of memory mapping functions in this volume of POSIX.1-2008 leads to the possibility that an application may allocate the synchronization objects from this section in memory that is accessed by multiple processes (and therefore, by threads of multiple processes).

In order to permit such usage, while at the same time keeping the usual case (that is, usage within a single process) efficient, a *process-shared* option has been defined.

If an implementation supports the `_POSIX_THREAD_PROCESS_SHARED` option, then the *process-shared* attribute can be used to indicate that mutexes or condition variables may be accessed by threads of multiple processes.

The default setting of `PTHREAD_PROCESS_PRIVATE` has been chosen for the *process-shared* attribute so that the most efficient forms of these synchronization objects are created by default.

Synchronization variables that are initialized with the `PTHREAD_PROCESS_PRIVATE` *process-shared* attribute may only be operated on by threads in the process that initialized them. Synchronization variables that are initialized with the `PTHREAD_PROCESS_SHARED` *process-shared* attribute may be operated on by any thread in any process that has access to it. In particular, these processes may exist beyond the lifetime of the initializing process. For example, the following code implements a simple counting semaphore in a mapped file that may be used by many processes.

```
/* sem.h */
struct semaphore {
    pthread_mutex_t lock;
    pthread_cond_t nonzero;
    unsigned count;
};
typedef struct semaphore semaphore_t;
```

```

53033 semaphore_t *semaphore_create(char *semaphore_name);
53034 semaphore_t *semaphore_open(char *semaphore_name);
53035 void semaphore_post(semaphore_t *semap);
53036 void semaphore_wait(semaphore_t *semap);
53037 void semaphore_close(semaphore_t *semap);

53038 /* sem.c */
53039 #include <sys/types.h>
53040 #include <sys/stat.h>
53041 #include <sys/mman.h>
53042 #include <fcntl.h>
53043 #include <pthread.h>
53044 #include "sem.h"

53045 semaphore_t *
53046 semaphore_create(char *semaphore_name)
53047 {
53048     int fd;
53049     semaphore_t *semap;
53050     pthread_mutexattr_t psharedm;
53051     pthread_condattr_t psharedc;

53052     fd = open(semaphore_name, O_RDWR | O_CREAT | O_EXCL, 0666);
53053     if (fd < 0)
53054         return (NULL);
53055     (void) ftruncate(fd, sizeof(semaphore_t));
53056     (void) pthread_mutexattr_init(&psharedm);
53057     (void) pthread_mutexattr_setpshared(&psharedm,
53058         PTHREAD_PROCESS_SHARED);
53059     (void) pthread_condattr_init(&psharedc);
53060     (void) pthread_condattr_setpshared(&psharedc,
53061         PTHREAD_PROCESS_SHARED);
53062     semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
53063         PROT_READ | PROT_WRITE, MAP_SHARED,
53064         fd, 0);
53065     close (fd);
53066     (void) pthread_mutex_init(&semap->lock, &psharedm);
53067     (void) pthread_cond_init(&semap->nonzero, &psharedc);
53068     semap->count = 0;
53069     return (semap);
53070 }

53071 semaphore_t *
53072 semaphore_open(char *semaphore_name)
53073 {
53074     int fd;
53075     semaphore_t *semap;

53076     fd = open(semaphore_name, O_RDWR, 0666);
53077     if (fd < 0)
53078         return (NULL);
53079     semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
53080         PROT_READ | PROT_WRITE, MAP_SHARED,
53081         fd, 0);

```

pthread_mutexattr_destroy()*System Interfaces*

```

53082         close (fd);
53083         return (semap);
53084     }

53085     void
53086     semaphore_post (semaphore_t *semap)
53087     {
53088         pthread_mutex_lock (&semap->lock);
53089         if (semap->count == 0)
53090             pthread_cond_signal (&semap->nonzero);
53091         semap->count++;
53092         pthread_mutex_unlock (&semap->lock);
53093     }

53094     void
53095     semaphore_wait (semaphore_t *semap)
53096     {
53097         pthread_mutex_lock (&semap->lock);
53098         while (semap->count == 0)
53099             pthread_cond_wait (&semap->nonzero, &semap->lock);
53100         semap->count--;
53101         pthread_mutex_unlock (&semap->lock);
53102     }

53103     void
53104     semaphore_close (semaphore_t *semap)
53105     {
53106         munmap((void *) semap, sizeof(semaphore_t));
53107     }

```

The following code is for three separate processes that create, post, and wait on a semaphore in the file **/tmp/semaphore**. Once the file is created, the post and wait programs increment and decrement the counting semaphore (waiting and waking as required) even though they did not initialize the semaphore.

```

53112     /* create.c */
53113     #include "pthread.h"
53114     #include "sem.h"

53115     int
53116     main()
53117     {
53118         semaphore_t *semap;

53119         semap = semaphore_create ("/tmp/semaphore");
53120         if (semap == NULL)
53121             exit(1);
53122         semaphore_close (semap);
53123         return (0);
53124     }

53125     /* post */
53126     #include "pthread.h"
53127     #include "sem.h"

53128     int

```

```

53129     main()
53130     {
53131         semaphore_t *semaph;
53132
53133         semaph = semaphore_open("/tmp/semaphore");
53134         if (semaph == NULL)
53135             exit(1);
53136         semaphore_post(semaph);
53137         semaphore_close(semaph);
53138         return (0);
53139     }
53140
53141     /* wait */
53142     #include "pthread.h"
53143     #include "sem.h"
53144
53145     int
53146     main()
53147     {
53148         semaphore_t *semaph;
53149
53150         semaph = semaphore_open("/tmp/semaphore");
53151         if (semaph == NULL)
53152             exit(1);
53153         semaphore_wait(semaph);
53154         semaphore_close(semaph);
53155         return (0);
53156     }

```

FUTURE DIRECTIONS

None.

SEE ALSO

[*pthread_cond_destroy\(\)*](#), [*pthread_create\(\)*](#), [*pthread_mutex_destroy\(\)*](#)

XBD [*<pthread.h>*](#)

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_mutexattr_destroy()* and *pthread_mutexattr_init()* functions are marked as part of the Threads option.

IEEE PASC Interpretation 1003.1c #27 is applied, updating the ERRORS section.

Issue 7

The *pthread_mutexattr_destroy()* and *pthread_mutexattr_init()* functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

pthread_mutexattr_getprioceiling()

System Interfaces

53169 **NAME**

53170 pthread_mutexattr_getprioceiling, pthread_mutexattr_setprioceiling — get and set the
 53171 prioceiling attribute of the mutex attributes object (**REALTIME THREADS**)

53172 **SYNOPSIS**

```
53173 RPP|TPP #include <pthread.h>
53174
53174 int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t
53175     *restrict attr, int *restrict prioceiling);
53176 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
53177     int prioceiling);
```

53178 **DESCRIPTION**

53179 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions,
 53180 respectively, shall get and set the priority ceiling attribute of a mutex attributes object pointed to
 53181 by *attr* which was previously created by the function *pthread_mutexattr_init()*.

53182 The *prioceiling* attribute contains the priority ceiling of initialized mutexes. The values of
 53183 *prioceiling* are within the maximum range of priorities defined by *SCHED_FIFO*.

53184 The *prioceiling* attribute defines the priority ceiling of initialized mutexes, which is the minimum
 53185 priority level at which the critical section guarded by the mutex is executed. In order to avoid
 53186 priority inversion, the priority ceiling of the mutex shall be set to a priority higher than or equal
 53187 to the highest priority of all the threads that may lock that mutex. The values of *prioceiling* are
 53188 within the maximum range of priorities defined under the *SCHED_FIFO* scheduling policy.

53189 The behavior is undefined if the value specified by the *attr* argument to
 53190 *pthread_mutexattr_getprioceiling()* or *pthread_mutexattr_setprioceiling()* does not refer to an
 53191 initialized mutex attributes object.

53192 **RETURN VALUE**

53193 Upon successful completion, the *pthread_mutexattr_getprioceiling()* and
 53194 *pthread_mutexattr_setprioceiling()* functions shall return zero; otherwise, an error number shall be
 53195 returned to indicate the error.

53196 **ERRORS**

53197 These functions may fail if:

53198 [EINVAL] The value specified by *prioceiling* is invalid.

53199 [EPERM] The caller does not have the privilege to perform the operation.

53200 These functions shall not return an error code of [EINTR].

53201 **EXAMPLES**

53202 None.

53203 **APPLICATION USAGE**

53204 None.

53205 **RATIONALE**

53206 If an implementation detects that the value specified by the *attr* argument to
 53207 *pthread_mutexattr_getprioceiling()* or *pthread_mutexattr_setprioceiling()* does not refer to an
 53208 initialized mutex attributes object, it is recommended that the function should fail and report an
 53209 [EINVAL] error.

53210 FUTURE DIRECTIONS

53211 None.

53212 SEE ALSO

53213 *pthread_cond_destroy()*, *pthread_create()*, *pthread_mutex_destroy()*

53214 XBD **<pthread.h>**

53215 CHANGE HISTORY

53216 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53217 Marked as part of the Realtime Threads Feature Group.

53218 Issue 6

53219 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions are marked
53220 as part of the Threads and Thread Priority Protection options.

53221 The [ENOSYS] error condition has been removed as stubs need not be provided if an
53222 implementation does not support the Thread Priority Protection option.

53223 The [ENOTSUP] error condition has been removed since these functions do not have a *protocol*
53224 argument.

53225 The **restrict** keyword is added to the *pthread_mutexattr_getprioceiling()* prototype for alignment
53226 with the ISO/IEC 9899:1999 standard.

53227 Issue 7

53228 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions are moved
53229 from the Threads option to require support of either the Robust Mutex Priority Protection option
53230 or the Non-Robust Mutex Priority Protection option.

53231 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition
53232 results in undefined behavior.

pthread_mutexattr_getprotocol()

System Interfaces

53233 **NAME**

53234 pthread_mutexattr_getprotocol, pthread_mutexattr_setprotocol — get and set the protocol
 53235 attribute of the mutex attributes object (**REALTIME THREADS**)

53236 **SYNOPSIS**

```
53237 MC1 #include <pthread.h>
53238
53238 int pthread_mutexattr_getprotocol(const pthread_mutexattr_t
53239     *restrict attr, int *restrict protocol);
53240 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
53241     int protocol);
```

53242 **DESCRIPTION**

53243 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions, respectively,
 53244 shall get and set the protocol attribute of a mutex attributes object pointed to by *attr* which was
 53245 previously created by the function *pthread_mutexattr_init()*.

53246 The *protocol* attribute defines the protocol to be followed in utilizing mutexes. The value of
 53247 *protocol* may be one of:

```
53248 RPI|TPI PTHREAD_PRIO_INHERIT
53249 MC1 PTHREAD_PRIO_NONE
53250 RPP|TPP PTHREAD_PRIO_PROTECT
```

53251 which are defined in the **<pthread.h>** header. The default value of the attribute shall be
 53252 PTHREAD_PRIO_NONE.

53253 When a thread owns a mutex with the PTHREAD_PRIO_NONE *protocol* attribute, its priority
 53254 and scheduling shall not be affected by its mutex ownership.

53255 RPI When a thread is blocking higher priority threads because of owning one or more robust
 53256 mutexes with the PTHREAD_PRIO_INHERIT *protocol* attribute, it shall execute at the higher of
 53257 its priority or the priority of the highest priority thread waiting on any of the robust mutexes
 53258 owned by this thread and initialized with this protocol.

53259 TPI When a thread is blocking higher priority threads because of owning one or more non-robust
 53260 mutexes with the PTHREAD_PRIO_INHERIT *protocol* attribute, it shall execute at the higher of
 53261 its priority or the priority of the highest priority thread waiting on any of the non-robust
 53262 mutexes owned by this thread and initialized with this protocol.

53263 RPP When a thread owns one or more robust mutexes initialized with the
 53264 PTHREAD_PRIO_PROTECT protocol, it shall execute at the higher of its priority or the highest
 53265 of the priority ceilings of all the robust mutexes owned by this thread and initialized with this
 53266 attribute, regardless of whether other threads are blocked on any of these robust mutexes or not.

53267 TPP When a thread owns one or more non-robust mutexes initialized with the
 53268 PTHREAD_PRIO_PROTECT protocol, it shall execute at the higher of its priority or the highest
 53269 of the priority ceilings of all the non-robust mutexes owned by this thread and initialized with
 53270 this attribute, regardless of whether other threads are blocked on any of these non-robust
 53271 mutexes or not.

53272 While a thread is holding a mutex which has been initialized with the
 53273 PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT protocol attributes, it shall not be
 53274 subject to being moved to the tail of the scheduling queue at its priority in the event that its
 53275 original priority is changed, such as by a call to *sched_setparam()*. Likewise, when a thread

53276 unlocks a mutex that has been initialized with the PTHREAD_PRIO_INHERIT or
 53277 PTHREAD_PRIO_PROTECT protocol attributes, it shall not be subject to being moved to the tail
 53278 of the scheduling queue at its priority in the event that its original priority is changed.

53279 If a thread simultaneously owns several mutexes initialized with different protocols, it shall
 53280 execute at the highest of the priorities that it would have obtained by each of these protocols.

53281 RPI|TPI When a thread makes a call to *pthread_mutex_lock()*, the mutex was initialized with the protocol
 53282 attribute having the value PTHREAD_PRIO_INHERIT, when the calling thread is blocked
 53283 because the mutex is owned by another thread, that owner thread shall inherit the priority level
 53284 of the calling thread as long as it continues to own the mutex. The implementation shall update
 53285 its execution priority to the maximum of its assigned priority and all its inherited priorities.
 53286 Furthermore, if this owner thread itself becomes blocked on another mutex with the *protocol*
 53287 attribute having the value PTHREAD_PRIO_INHERIT, the same priority inheritance effect shall
 53288 be propagated to this other owner thread, in a recursive manner.

53289 The behavior is undefined if the value specified by the *attr* argument to
 53290 *pthread_mutexattr_getprotocol()* or *pthread_mutexattr_setprotocol()* does not refer to an initialized
 53291 mutex attributes object.

53292 RETURN VALUE

53293 Upon successful completion, the *pthread_mutexattr_getprotocol()* and
 53294 *pthread_mutexattr_setprotocol()* functions shall return zero; otherwise, an error number shall be
 53295 returned to indicate the error.

53296 ERRORS

53297 The *pthread_mutexattr_setprotocol()* function shall fail if:

53298 [ENOTSUP] The value specified by *protocol* is an unsupported value.

53299 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions may fail if:

53300 [EINVAL] The value specified by *protocol* is invalid.

53301 [EPERM] The caller does not have the privilege to perform the operation.

53302 These functions shall not return an error code of [EINTR].

53303 EXAMPLES

53304 None.

53305 APPLICATION USAGE

53306 None.

53307 RATIONALE

53308 If an implementation detects that the value specified by the *attr* argument to
 53309 *pthread_mutexattr_getprotocol()* or *pthread_mutexattr_setprotocol()* does not refer to an initialized
 53310 mutex attributes object, it is recommended that the function should fail and report an [EINVAL]
 53311 error.

53312 FUTURE DIRECTIONS

53313 None.

53314 SEE ALSO

53315 *pthread_cond_destroy()*, *pthread_create()*, *pthread_mutex_destroy()*

53316 XBD [*<pthread.h>*](#)

pthread_mutexattr_getprotocol()

System Interfaces

CHANGE HISTORY

53317 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53318 Marked as part of the Realtime Threads Feature Group.

Issue 6

53320 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions are marked as
53321 part of the Threads option and either the Thread Priority Protection or Thread Priority
53322 Inheritance options.

53324 The [ENOSYS] error condition has been removed as stubs need not be provided if an
53325 implementation does not support the Thread Priority Protection or Thread Priority Inheritance
53326 options.

53327 The **restrict** keyword is added to the *pthread_mutexattr_getprotocol()* prototype for alignment
53328 with the ISO/IEC 9899:1999 standard.

Issue 7

53329 SD5-XSH-ERN-135 is applied, updating the DESCRIPTION to define a default value for the
53330 *protocol* attribute.

53331 SD5-XSH-ERN-188 is applied, updating the DESCRIPTION.

53332 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions are moved from
53333 the Threads option to require support of either the Non-Robust Mutex Priority Protection option
53334 or the Non-Robust Mutex Priority Inheritance option or the Robust Mutex Priority Protection
53335 option or the Robust Mutex Priority Inheritance option.

53336 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition
53337 results in undefined behavior.

NAME

pthread_mutexattr_getpshared, pthread_mutexattr_setpshared — get and set the process-shared attribute

SYNOPSIS

```
TSH #include <pthread.h>

int pthread_mutexattr_getpshared(const pthread_mutexattr_t
    *restrict attr, int *restrict pshared);
int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_mutexattr_getpshared()* function shall obtain the value of the *process-shared* attribute from the attributes object referenced by *attr*.

The *pthread_mutexattr_setpshared()* function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute is set to `PTHREAD_PROCESS_SHARED` to permit a mutex to be operated upon by any thread that has access to the memory where the mutex is allocated, even if the mutex is allocated in memory that is shared by multiple processes. If the *process-shared* attribute is `PTHREAD_PROCESS_PRIVATE`, the mutex shall only be operated upon by threads created within the same process as the thread that initialized the mutex; if threads of differing processes attempt to operate on such a mutex, the behavior is undefined. The default value of the attribute shall be `PTHREAD_PROCESS_PRIVATE`.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_getpshared()* or *pthread_mutexattr_setpshared()* does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, *pthread_mutexattr_setpshared()* shall return zero; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *pthread_mutexattr_getpshared()* shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_setpshared()* function may fail if:

[EINVAL] The new value specified for the attribute is outside the range of legal values for that attribute.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_mutexattr_getpshared()* or *pthread_mutexattr_setpshared()* does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an [EINVAL] error.

pthread_mutexattr_getpshared()

System Interfaces

53383 **FUTURE DIRECTIONS**

53384 None.

53385 **SEE ALSO**53386 *pthread_cond_destroy()*, *pthread_create()*, *pthread_mutex_destroy()*, *pthread_mutexattr_destroy()*53387 XBD **<pthread.h>**53388 **CHANGE HISTORY**

53389 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53390 **Issue 6**53391 The *pthread_mutexattr_getpshared()* and *pthread_mutexattr_setpshared()* functions are marked as
53392 part of the Threads and Thread Process-Shared Synchronization options.53393 The **restrict** keyword is added to the *pthread_mutexattr_getpshared()* prototype for alignment
53394 with the ISO/IEC 9899:1999 standard.53395 **Issue 7**53396 The *pthread_mutexattr_getpshared()* and *pthread_mutexattr_setpshared()* functions are moved from
53397 the Threads option.53398 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition
53399 results in undefined behavior.

System Interfaces

pthread_mutexattr_getrobust()**NAME**

pthread_mutexattr_getrobust, pthread_mutexattr_setrobust — get and set the mutex robust attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict
    attr, int *restrict robust);
int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
    int robust);
```

DESCRIPTION

The *pthread_mutexattr_getrobust()* and *pthread_mutexattr_setrobust()* functions, respectively, shall get and set the mutex *robust* attribute. This attribute is set in the *robust* parameter. Valid values for *robust* include:

PTHREAD_MUTEX_STALLED

No special actions are taken if the owner of the mutex is terminated while holding the mutex lock. This can lead to deadlocks if no other thread can unlock the mutex. This is the default value.

PTHREAD_MUTEX_ROBUST

If the process containing the owning thread of a robust mutex terminates while holding the mutex lock, the next thread that acquires the mutex shall be notified about the termination by the return value [EOWNERDEAD] from the locking function. If the owning thread of a robust mutex terminates while holding the mutex lock, the next thread that acquires the mutex may be notified about the termination by the return value [EOWNERDEAD]. The notified thread can then attempt to mark the state protected by the mutex as consistent again by a call to *pthread_mutex_consistent()*. After a subsequent successful call to *pthread_mutex_unlock()*, the mutex lock shall be released and can be used normally by other threads. If the mutex is unlocked without a call to *pthread_mutex_consistent()*, it shall be in a permanently unusable state and all attempts to lock the mutex shall fail with the error [ENOTRECOVERABLE]. The only permissible operation on such a mutex is *pthread_mutex_destroy()*.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_getrobust()* or *pthread_mutexattr_setrobust()* does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, the *pthread_mutexattr_getrobust()* function shall return zero and store the value of the *robust* attribute of *attr* into the object referenced by the *robust* parameter. Otherwise, an error value shall be returned to indicate the error. If successful, the *pthread_mutexattr_setrobust()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_setrobust()* function shall fail if:

[EINVAL] The value of *robust* is invalid.

These functions shall not return an error code of [EINTR].

pthread_mutexattr_getrobust()

System Interfaces

53443 **EXAMPLES**

53444 None.

53445 **APPLICATION USAGE**

53446 The actions required to make the state protected by the mutex consistent again are solely
53447 dependent on the application. If it is not possible to make the state of a mutex consistent, robust
53448 mutexes can be used to notify this situation by calling *pthread_mutex_unlock()* without a prior
53449 call to *pthread_mutex_consistent()*.

53450 If the state is declared inconsistent by calling *pthread_mutex_unlock()* without a prior call to
53451 *pthread_mutex_consistent()*, a possible approach could be to destroy the mutex and then
53452 reinitialize it. However, it should be noted that this is possible only in certain situations where
53453 the state protected by the mutex has to be reinitialized and coordination achieved with other
53454 threads blocked on the mutex, because otherwise a call to a locking function with a reference to a
53455 mutex object invalidated by a call to *pthread_mutex_destroy()* results in undefined behavior.

53456 **RATIONALE**

53457 If an implementation detects that the value specified by the *attr* argument to
53458 *pthread_mutexattr_getrobust()* or *pthread_mutexattr_setrobust()* does not refer to an initialized
53459 mutex attributes object, it is recommended that the function should fail and report an [EINVAL]
53460 error.

53461 **FUTURE DIRECTIONS**

53462 None.

53463 **SEE ALSO**53464 *pthread_mutex_consistent()*, *pthread_mutex_destroy()*, *pthread_mutex_lock()*

53465 XBD <pthread.h>

53466 **CHANGE HISTORY**

53467 First released in Issue 7.

NAME

pthread_mutexattr_gettype, pthread_mutexattr_settype — get and set the mutex type attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
    int *restrict type);
int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

DESCRIPTION

The *pthread_mutexattr_gettype()* and *pthread_mutexattr_settype()* functions, respectively, shall get and set the mutex *type* attribute. This attribute is set in the *type* parameter to these functions. The default value of the *type* attribute is PTHREAD_MUTEX_DEFAULT.

The type of mutex is contained in the *type* attribute of the mutex attributes. Valid mutex types include:

PTHREAD_MUTEX_NORMAL

This type of mutex does not detect deadlock. A thread attempting to relock this mutex without first unlocking it shall deadlock. Attempting to unlock a mutex locked by a different thread results in undefined behavior. Attempting to unlock an unlocked mutex results in undefined behavior.

PTHREAD_MUTEX_ERRORCHECK

This type of mutex provides error checking. A thread attempting to relock this mutex without first unlocking it shall return with an error. A thread attempting to unlock a mutex which another thread has locked shall return with an error. A thread attempting to unlock an unlocked mutex shall return with an error.

PTHREAD_MUTEX_RECURSIVE

A thread attempting to relock this mutex without first unlocking it shall succeed in locking the mutex. The relocking deadlock which can occur with mutexes of type PTHREAD_MUTEX_NORMAL cannot occur with this type of mutex. Multiple locks of this mutex shall require the same number of unlocks to release the mutex before another thread can acquire the mutex. A thread attempting to unlock a mutex which another thread has locked shall return with an error. A thread attempting to unlock an unlocked mutex shall return with an error.

PTHREAD_MUTEX_DEFAULT

Attempting to recursively lock a mutex of this type results in undefined behavior. Attempting to unlock a mutex of this type which was not locked by the calling thread results in undefined behavior. Attempting to unlock a mutex of this type which is not locked results in undefined behavior. An implementation may map this mutex to one of the other mutex types.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_gettype()* or *pthread_mutexattr_settype()* does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, the *pthread_mutexattr_gettype()* function shall return zero and store the value of the *type* attribute of *attr* into the object referenced by the *type* parameter. Otherwise, an error shall be returned to indicate the error.

If successful, the *pthread_mutexattr_settype()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

pthread_mutexattr_gettype()

System Interfaces

53514 ERRORS

53515 The *pthread_mutexattr_settype()* function shall fail if:

53516 [EINVAL] The value *type* is invalid.

53517 These functions shall not return an error code of [EINTR].

53518 EXAMPLES

53519 None.

53520 APPLICATION USAGE

53521 It is advised that an application should not use a PTHREAD_MUTEX_RECURSIVE mutex with
 53522 condition variables because the implicit unlock performed for a *pthread_cond_timedwait()* or
 53523 *pthread_cond_wait()* may not actually release the mutex (if it had been locked multiple times). If
 53524 this happens, no other thread can satisfy the condition of the predicate.

53525 RATIONALE

53526 If an implementation detects that the value specified by the *attr* argument to
 53527 *pthread_mutexattr_gettype()* or *pthread_mutexattr_settype()* does not refer to an initialized mutex
 53528 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

53529 FUTURE DIRECTIONS

53530 None.

53531 SEE ALSO

53532 *pthread_cond_timedwait()*

53533 XBD <pthread.h>

53534 CHANGE HISTORY

53535 First released in Issue 5.

53536 Issue 6

53537 The Open Group Corrigendum U033/3 is applied. The SYNOPSIS for
 53538 *pthread_mutexattr_gettype()* is updated so that the first argument is of type **const**
 53539 **pthread_mutexattr_t***.

53540 The **restrict** keyword is added to the *pthread_mutexattr_gettype()* prototype for alignment with
 53541 the ISO/IEC 9899:1999 standard.

53542 Issue 7

53543 The *pthread_mutexattr_gettype()* and *pthread_mutexattr_settype()* functions are moved from the
 53544 XSI option to the Base.

53545 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition
 53546 results in undefined behavior.

*System Interfaces***pthread_mutexattr_init()**53547 **NAME**

53548 pthread_mutexattr_init — initialize the mutex attributes object

53549 **SYNOPSIS**

53550 #include <pthread.h>

53551 int pthread_mutexattr_init(pthread_mutexattr_t *attr);

53552 **DESCRIPTION**53553 Refer to *pthread_mutexattr_destroy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_mutexattr_setprioceiling()*System Interfaces*53554 **NAME**

53555 pthread_mutexattr_setprioceiling — set the prioceiling attribute of the mutex attributes object
53556 (**REALTIME THREADS**)

53557 **SYNOPSIS**

```
53558 RPP|TPP #include <pthread.h>  
53559  
53559 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,  
53560 int prioceiling);
```

53561 **DESCRIPTION**

53562 Refer to *pthread_mutexattr_getprioceiling()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread_mutexattr_setprotocol()**53563 **NAME**

53564 pthread_mutexattr_setprotocol — set the protocol attribute of the mutex attributes object
53565 (**REALTIME THREADS**)

53566 **SYNOPSIS**

```
53567 MC1 #include <pthread.h>  
53568      int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,  
53569      int protocol);
```

53570 **DESCRIPTION**

53571 Refer to [pthread_mutexattr_getprotocol\(\)](#).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_mutexattr_setpshared()*System Interfaces*53572 **NAME**

53573 pthread_mutexattr_setpshared — set the process-shared attribute

53574 **SYNOPSIS**

```
53575 TSH #include <pthread.h>  
53576 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,  
53577 int pshared);
```

53578 **DESCRIPTION**53579 Refer to *pthread_mutexattr_getpshared()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread_mutexattr_setrobust()**53580 **NAME**

53581 pthread_mutexattr_setrobust — get and set the mutex robust attribute

53582 **SYNOPSIS**

53583 #include <pthread.h>

53584 int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
53585 int robust);53586 **DESCRIPTION**53587 Refer to *pthread_mutexattr_getrobust()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_mutexattr_settype()*System Interfaces*53588 **NAME**

53589 pthread_mutexattr_settype — set the mutex type attribute

53590 **SYNOPSIS**

53591 #include <pthread.h>

53592 int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);

53593 **DESCRIPTION**53594 Refer to *pthread_mutexattr_gettype()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

pthread_once — dynamic package initialization

SYNOPSIS

```
#include <pthread.h>

int pthread_once(pthread_once_t *once_control,
void (*init_routine)(void));

pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

DESCRIPTION

The first call to *pthread_once()* by any thread in a process, with a given *once_control*, shall call the *init_routine* with no arguments. Subsequent calls of *pthread_once()* with the same *once_control* shall not call the *init_routine*. On return from *pthread_once()*, *init_routine* shall have completed. The *once_control* parameter shall determine whether the associated initialization routine has been called.

The *pthread_once()* function is not a cancellation point. However, if *init_routine* is a cancellation point and is canceled, the effect on *once_control* shall be as if *pthread_once()* was never called.

The constant PTHREAD_ONCE_INIT is defined in the **<pthread.h>** header.

The behavior of *pthread_once()* is undefined if *once_control* has automatic storage duration or is not initialized by PTHREAD_ONCE_INIT.

RETURN VALUE

Upon successful completion, *pthread_once()* shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_once()* function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Some C libraries are designed for dynamic initialization. That is, the global initialization for the library is performed when the first procedure in the library is called. In a single-threaded program, this is normally implemented using a static variable whose value is checked on entry to a routine, as follows:

```
static int random_is_initialized = 0;
extern int initialize_random();

int random_function()
{
    if (random_is_initialized == 0) {
        initialize_random();
        random_is_initialized = 1;
    }
    ... /* Operations performed after initialization. */
}
```

To keep the same structure in a multi-threaded program, a new primitive is needed. Otherwise, library initialization has to be accomplished by an explicit call to a library-exported initialization function prior to any use of the library.

pthread_once()*System Interfaces*

For dynamic library initialization in a multi-threaded process, a simple initialization flag is not sufficient; the flag needs to be protected against modification by multiple threads simultaneously calling into the library. Protecting the flag requires the use of a mutex; however, mutexes have to be initialized before they are used. Ensuring that the mutex is only initialized once requires a recursive solution to this problem.

The use of *pthread_once()* not only supplies an implementation-guaranteed means of dynamic initialization, it provides an aid to the reliable construction of multi-threaded and realtime systems. The preceding example then becomes:

```
#include <pthread.h>
static pthread_once_t random_is_initialized = PTHREAD_ONCE_INIT;
extern int initialize_random();

int random_function()
{
    (void) pthread_once(&random_is_initialized, initialize_random);
    ... /* Operations performed after initialization. */
}
```

Note that a **pthread_once_t** cannot be an array because some compilers do not accept the construct **&<array_name>**.

If an implementation detects that the value specified by the *once_control* argument to *pthread_once()* does not refer to a **pthread_once_t** object initialized by PTHREAD_ONCE_INIT, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD [<pthread.h>](#)

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_once()* function is marked as part of the Threads option.

The [EINVAL] error is added as a “may fail” case for if either argument is invalid.

Issue 7

The *pthread_once()* function is moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized **pthread_once_t** object is removed; this condition results in undefined behavior.

NAME

pthread_rwlock_destroy, pthread_rwlock_init — destroy and initialize a read-write lock object

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
    const pthread_rwlockattr_t *restrict attr);
pthread_rwlock_t rwlock = PTHREAD_RWLOCK_INITIALIZER;
```

DESCRIPTION

The *pthread_rwlock_destroy()* function shall destroy the read-write lock object referenced by *rwlock* and release any resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is reinitialized by another call to *pthread_rwlock_init()*. An implementation may cause *pthread_rwlock_destroy()* to set the object referenced by *rwlock* to an invalid value. Results are undefined if *pthread_rwlock_destroy()* is called when any thread holds *rwlock*. Attempting to destroy an uninitialized read-write lock results in undefined behavior.

The *pthread_rwlock_init()* function shall allocate any resources required to use the read-write lock referenced by *rwlock* and initializes the lock to an unlocked state with attributes referenced by *attr*. If *attr* is NULL, the default read-write lock attributes shall be used; the effect is the same as passing the address of a default read-write lock attributes object. Once initialized, the lock can be used any number of times without being reinitialized. Results are undefined if *pthread_rwlock_init()* is called specifying an already initialized read-write lock. Results are undefined if a read-write lock is used without first being initialized.

If the *pthread_rwlock_init()* function fails, *rwlock* shall not be initialized and the contents of *rwlock* are undefined.

Only the object referenced by *rwlock* may be used for performing synchronization. The result of referring to copies of that object in calls to *pthread_rwlock_destroy()*, *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlock_tryrdlock()*, *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*, or *pthread_rwlock_wrlock()* is undefined.

In cases where default read-write lock attributes are appropriate, the macro PTHREAD_RWLOCK_INITIALIZER can be used to initialize read-write locks that are statically allocated. The effect shall be equivalent to dynamic initialization by a call to *pthread_rwlock_init()* with the *attr* parameter specified as NULL, except that no error checks are performed.

The behavior is undefined if the value specified by the *attr* argument to *pthread_rwlock_init()* does not refer to an initialized read-write lock attributes object.

RETURN VALUE

If successful, the *pthread_rwlock_destroy()* and *pthread_rwlock_init()* functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_init()* function shall fail if:

[EAGAIN] The system lacked the necessary resources (other than memory) to initialize another read-write lock.

[ENOMEM] Insufficient memory exists to initialize the read-write lock.

[EPERM] The caller does not have the privilege to perform the operation.

These functions shall not return an error code of [EINTR].

pthread_rwlock_destroy()

System Interfaces

EXAMPLES

None.

APPLICATION USAGE

Applications using these and related read-write lock functions may be subject to priority inversion, as discussed in XBD [Section 3.285](#) (on page 79).

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_destroy()* does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *attr* argument to *pthread_rwlock_t_init()* does not refer to an initialized read-write lock attributes object, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_destroy()* or *pthread_rwlock_init()* refers to a locked read-write lock object, or detects that the value specified by the *rwlock* argument to *pthread_rwlock_init()* refers to an already initialized read-write lock object, it is recommended that the function should fail and report an [EBUSY] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_rdlock(), *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*

XBD [Section 3.285](#) (on page 79), [<pthread.h>](#)

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension). The initializer macro is also deleted from the SYNOPSIS.
- The DESCRIPTION is updated as follows:
 - It explicitly notes allocation of resources upon initialization of a read-write lock object.
 - A paragraph is added specifying that copies of read-write lock objects may not be used.
- An [EINVAL] error is added to the ERRORS section for *pthread_rwlock_init()*, indicating that the *rwlock* value is invalid.
- The SEE ALSO section is updated.

The **restrict** keyword is added to the *pthread_rwlock_init()* prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/45 is applied, adding APPLICATION USAGE relating to priority inversion.

53761 **Issue 7**

53762 Austin Group Interpretation 1003.1-2001 #048 is applied, adding the
53763 PTHREAD_RWLOCK_INITIALIZER macro.

53764 The *pthread_rwlock_destroy()* and *pthread_rwlock_init()* functions are moved from the Threads
53765 option to the Base.

53766 The [EINVAL] error for an uninitialized read-write lock object or read-write lock attributes
53767 object is removed; this condition results in undefined behavior.

53768 The [EBUSY] error for a locked read-write lock object or an already initialized read-write lock
53769 object is removed; this condition results in undefined behavior.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_rwlock_rdlock()

System Interfaces

53770 NAME

53771 pthread_rwlock_rdlock, pthread_rwlock_tryrdlock — lock a read-write lock object for reading

53772 SYNOPSIS

53773 #include <pthread.h>

53774 int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);

53775 int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);

53776 DESCRIPTION

53777 The *pthread_rwlock_rdlock()* function shall apply a read lock to the read-write lock referenced by
 53778 *rwlock*. The calling thread acquires the read lock if a writer does not hold the lock and there are
 53779 no writers blocked on the lock.

53780 TPS If the Thread Execution Scheduling option is supported, and the threads involved in the lock are
 53781 executing with the scheduling policies SCHED_FIFO or SCHED_RR, the calling thread shall not
 53782 acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on
 53783 the lock; otherwise, the calling thread shall acquire the lock.

53784 TPS TSP If the Thread Execution Scheduling option is supported, and the threads involved in the lock are
 53785 executing with the SCHED_SPORADIC scheduling policy, the calling thread shall not acquire
 53786 the lock if a writer holds the lock or if writers of higher or equal priority are blocked on the lock;
 53787 otherwise, the calling thread shall acquire the lock.

53788 If the Thread Execution Scheduling option is not supported, it is implementation-defined
 53789 whether the calling thread acquires the lock when a writer does not hold the lock and there are
 53790 writers blocked on the lock. If a writer holds the lock, the calling thread shall not acquire the
 53791 read lock. If the read lock is not acquired, the calling thread shall block until it can acquire the
 53792 lock. The calling thread may deadlock if at the time the call is made it holds a write lock.

53793 A thread may hold multiple concurrent read locks on *rwlock* (that is, successfully call the
 53794 *pthread_rwlock_rdlock()* function *n* times). If so, the application shall ensure that the thread
 53795 performs matching unlocks (that is, it calls the *pthread_rwlock_unlock()* function *n* times).

53796 The maximum number of simultaneous read locks that an implementation guarantees can be
 53797 applied to a read-write lock shall be implementation-defined. The *pthread_rwlock_rdlock()*
 53798 function may fail if this maximum would be exceeded.

53799 The *pthread_rwlock_tryrdlock()* function shall apply a read lock as in the *pthread_rwlock_rdlock()*
 53800 function, with the exception that the function shall fail if the equivalent *pthread_rwlock_rdlock()*
 53801 call would have blocked the calling thread. In no case shall the *pthread_rwlock_tryrdlock()*
 53802 function ever block; it always either acquires the lock or fails and returns immediately.

53803 Results are undefined if any of these functions are called with an uninitialized read-write lock.

53804 If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the
 53805 signal handler the thread resumes waiting for the read-write lock for reading as if it was not
 53806 interrupted.

53807 RETURN VALUE

53808 If successful, the *pthread_rwlock_rdlock()* function shall return zero; otherwise, an error number
 53809 shall be returned to indicate the error.

53810 The *pthread_rwlock_tryrdlock()* function shall return zero if the lock for reading on the read-write
 53811 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to
 53812 indicate the error.

ERRORS

The *pthread_rwlock_tryrdlock()* function shall fail if:

[EBUSY] The read-write lock could not be acquired for reading because a writer holds the lock or a writer with the appropriate priority was blocked on it.

The *pthread_rwlock_rdlock()* and *pthread_rwlock_tryrdlock()* functions may fail if:

[EAGAIN] The read lock could not be acquired because the maximum number of read locks for *rwlock* has been exceeded.

The *pthread_rwlock_rdlock()* function may fail if:

[EDEADLK] A deadlock condition was detected or the current thread already owns the read-write lock for writing.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.285 (on page 79).

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_rdlock()* or *pthread_rwlock_tryrdlock()* does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy(), *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*

XBD Section 3.285 (on page 79), Section 4.11 (on page 110), **<pthread.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The DESCRIPTION is updated as follows:
 - Conditions under which writers have precedence over readers are specified.
 - Failure of *pthread_rwlock_tryrdlock()* is clarified.
 - A paragraph on the maximum number of read locks is added.
- In the ERRORS sections, [EBUSY] is modified to take into account write priority, and [EDEADLK] is deleted as a *pthread_rwlock_tryrdlock()* error.

pthread_rwlock_rdlock()*System Interfaces*

- 53852 • The SEE ALSO section is updated.
- 53853 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/101 is applied, updating the ERRORS
- 53854 section so that the [EDEADLK] error includes detection of a deadlock condition.
- 53855 **Issue 7**
- 53856 The *pthread_rwlock_rdlock()* and *pthread_rwlock_tryrdlock()* functions are moved from the Threads
- 53857 option to the Base.
- 53858 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results
- 53859 in undefined behavior.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

pthread_rwlock_timedrdlock — lock a read-write lock for reading

SYNOPSIS

```
#include <pthread.h>
#include <time.h>
```

```
int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abstime);
```

DESCRIPTION

The *pthread_rwlock_timedrdlock()* function shall apply a read lock to the read-write lock referenced by *rwlock* as in the *pthread_rwlock_rdlock()* function. However, if the lock cannot be acquired without waiting for other threads to unlock the lock, this wait shall be terminated when the specified timeout expires. The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the CLOCK_REALTIME clock. The **timespec** data type is defined in the **<time.h>** header. Under no circumstances shall the function fail with a timeout if the lock can be acquired immediately. The validity of the *abstime* parameter need not be checked if the lock can be immediately acquired.

If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-write lock via a call to *pthread_rwlock_timedrdlock()*, upon return from the signal handler the thread shall resume waiting for the lock as if it was not interrupted.

The calling thread may deadlock if at the time the call is made it holds a write lock on *rwlock*. The results are undefined if this function is called with an uninitialized read-write lock.

RETURN VALUE

The *pthread_rwlock_timedrdlock()* function shall return zero if the lock for reading on the read-write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_timedrdlock()* function shall fail if:

[ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

The *pthread_rwlock_timedrdlock()* function may fail if:

[EAGAIN] The read lock could not be acquired because the maximum number of read locks for lock would be exceeded.

[EDEADLK] A deadlock condition was detected or the calling thread already holds a write lock on *rwlock*.

[EINVAL] The *abstime* nanosecond value is less than zero or greater than or equal to 1 000 million.

This function shall not return an error code of [EINTR].

pthread_rwlock_timedrdlock()

System Interfaces

53900 EXAMPLES

53901 None.

53902 APPLICATION USAGE

53903 Applications using this function may be subject to priority inversion, as discussed in XBD
 53904 [Section 3.285](#) (on page 79).

53905 RATIONALE

53906 If an implementation detects that the value specified by the *rwlock* argument to
 53907 *pthread_rwlock_timedrdlock()* does not refer to an initialized read-write lock object, it is
 53908 recommended that the function should fail and report an [EINVAL] error.

53909 FUTURE DIRECTIONS

53910 None.

53911 SEE ALSO

53912 [pthread_rwlock_destroy\(\)](#), [pthread_rwlock_rdlock\(\)](#), [pthread_rwlock_timedwrlock\(\)](#),
 53913 [pthread_rwlock_trywrlock\(\)](#), [pthread_rwlock_unlock\(\)](#)
 53914 XBD [Section 3.285](#) (on page 79), [Section 4.11](#) (on page 110), [<pthread.h>](#), [<time.h>](#)

53915 CHANGE HISTORY

53916 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

53917 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/102 is applied, updating the ERRORS
 53918 section so that the [EDEADLK] error includes detection of a deadlock condition.

53919 Issue 753920 The *pthread_rwlock_timedrdlock()* function is moved from the Timeouts option to the Base.

53921 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results
 53922 in undefined behavior.

NAME

pthread_rwlock_timedwrlock — lock a read-write lock for writing

SYNOPSIS

```
#include <pthread.h>
#include <time.h>

int pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abstime);
```

DESCRIPTION

The *pthread_rwlock_timedwrlock()* function shall apply a write lock to the read-write lock referenced by *rwlock* as in the *pthread_rwlock_wrlock()* function. However, if the lock cannot be acquired without waiting for other threads to unlock the lock, this wait shall be terminated when the specified timeout expires. The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the CLOCK_REALTIME clock. The **timespec** data type is defined in the **<time.h>** header. Under no circumstances shall the function fail with a timeout if the lock can be acquired immediately. The validity of the *abstime* parameter need not be checked if the lock can be immediately acquired.

If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-write lock via a call to *pthread_rwlock_timedwrlock()*, upon return from the signal handler the thread shall resume waiting for the lock as if it was not interrupted.

The calling thread may deadlock if at the time the call is made it holds the read-write lock. The results are undefined if this function is called with an uninitialized read-write lock.

RETURN VALUE

The *pthread_rwlock_timedwrlock()* function shall return zero if the lock for writing on the read-write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_timedwrlock()* function shall fail if:

[ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

The *pthread_rwlock_timedwrlock()* function may fail if:

[EDEADLK] A deadlock condition was detected or the calling thread already holds the *rwlock*.

[EINVAL] The *abstime* nanosecond value is less than zero or greater than or equal to 1 000 million.

This function shall not return an error code of [EINTR].

pthread_rwlock_timedwrlock()

System Interfaces

53961 **EXAMPLES**

53962 None.

53963 **APPLICATION USAGE**

53964 Applications using this function may be subject to priority inversion, as discussed in XBD
 53965 [Section 3.285](#) (on page 79).

53966 **RATIONALE**

53967 If an implementation detects that the value specified by the *rwlock* argument to
 53968 *pthread_rwlock_timedwrlock()* does not refer to an initialized read-write lock object, it is
 53969 recommended that the function should fail and report an [EINVAL] error.

53970 **FUTURE DIRECTIONS**

53971 None.

53972 **SEE ALSO**

53973 [pthread_rwlock_destroy\(\)](#), [pthread_rwlock_rdlock\(\)](#), [pthread_rwlock_timedrdlock\(\)](#),
 53974 [pthread_rwlock_trywrlock\(\)](#), [pthread_rwlock_unlock\(\)](#)

53975 XBD [Section 3.285](#) (on page 79), [Section 4.11](#) (on page 110), [<pthread.h>](#), [<time.h>](#)

53976 **CHANGE HISTORY**

53977 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

53978 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/103 is applied, updating the ERRORS
 53979 section so that the [EDEADLK] error includes detection of a deadlock condition.

53980 **Issue 7**

53981 The *pthread_rwlock_timedwrlock()* function is moved from the Timeouts option to the Base.

53982 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results
 53983 in undefined behavior.

*System Interfaces***pthread_rwlock_tryrdlock()**53984 **NAME**

53985 pthread_rwlock_tryrdlock — lock a read-write lock object for reading

53986 **SYNOPSIS**

53987 #include <pthread.h>

53988 int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);

53989 **DESCRIPTION**53990 Refer to *pthread_rwlock_rdlock()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_rwlock_trywrlock()

System Interfaces

53991 NAME

53992 pthread_rwlock_trywrlock, pthread_rwlock_wrlock — lock a read-write lock object for writing

53993 SYNOPSIS

53994 #include <pthread.h>

53995 int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);

53996 int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);

53997 DESCRIPTION

53998 The *pthread_rwlock_trywrlock()* function shall apply a write lock like the *pthread_rwlock_wrlock()*
 53999 function, with the exception that the function shall fail if any thread currently holds *rwlock* (for
 54000 reading or writing).

54001 The *pthread_rwlock_wrlock()* function shall apply a write lock to the read-write lock referenced by
 54002 *rwlock*. The calling thread acquires the write lock if no other thread (reader or writer) holds the
 54003 read-write lock *rwlock*. Otherwise, the thread shall block until it can acquire the lock. The calling
 54004 thread may deadlock if at the time the call is made it holds the read-write lock (whether a read
 54005 or write lock).

54006 Implementations may favor writers over readers to avoid writer starvation.

54007 Results are undefined if any of these functions are called with an uninitialized read-write lock.

54008 If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the
 54009 signal handler the thread resumes waiting for the read-write lock for writing as if it was not
 54010 interrupted.

54011 RETURN VALUE

54012 The *pthread_rwlock_trywrlock()* function shall return zero if the lock for writing on the read-write
 54013 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to
 54014 indicate the error.

54015 If successful, the *pthread_rwlock_wrlock()* function shall return zero; otherwise, an error number
 54016 shall be returned to indicate the error.

54017 ERRORS

54018 The *pthread_rwlock_trywrlock()* function shall fail if:

54019 [EBUSY] The read-write lock could not be acquired for writing because it was already
 54020 locked for reading or writing.

54021 The *pthread_rwlock_wrlock()* function may fail if:

54022 [EDEADLK] A deadlock condition was detected or the current thread already owns the
 54023 read-write lock for writing or reading.

54024 These functions shall not return an error code of [EINTR].

54025 EXAMPLES

54026 None.

54027 APPLICATION USAGE

54028 Applications using these functions may be subject to priority inversion, as discussed in XBD
 54029 [Section 3.285](#) (on page 79).

54030 RATIONALE

54031 If an implementation detects that the value specified by the *rwlock* argument to
 54032 *pthread_rwlock_trywrlock()* or *pthread_rwlock_wrlock()* does not refer to an initialized read-write
 54033 lock object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy(), *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*,
pthread_rwlock_timedwrlock(), *pthread_rwlock_unlock()*

XBD [Section 3.285](#) (on page 79), [Section 4.11](#) (on page 110), [<pthread.h>](#)

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The [EDEADLK] error is deleted as a *pthread_rwlock_trywrlock()* error.
- The SEE ALSO section is updated.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/104 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

Issue 7

The *pthread_rwlock_trywrlock()* and *pthread_rwlock_wrlock()* functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

pthread_rwlock_unlock()

System Interfaces

54056 **NAME**

54057 pthread_rwlock_unlock — unlock a read-write lock object

54058 **SYNOPSIS**

54059 #include <pthread.h>

54060 int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);

54061 **DESCRIPTION**

54062 The *pthread_rwlock_unlock()* function shall release a lock held on the read-write lock object
 54063 referenced by *rwlock*. Results are undefined if the read-write lock *rwlock* is not held by the
 54064 calling thread.

54065 If this function is called to release a read lock from the read-write lock object and there are other
 54066 read locks currently held on this read-write lock object, the read-write lock object remains in the
 54067 read locked state. If this function releases the last read lock for this read-write lock object, the
 54068 read-write lock object shall be put in the unlocked state with no owners.

54069 If this function is called to release a write lock for this read-write lock object, the read-write lock
 54070 object shall be put in the unlocked state.

54071 If there are threads blocked on the lock when it becomes available, the scheduling policy shall
 54072 determine which thread(s) shall acquire the lock. If the Thread Execution Scheduling option is
 54073 supported, when threads executing with the scheduling policies SCHED_FIFO, SCHED_RR, or
 54074 SCHED_SPORADIC are waiting on the lock, they shall acquire the lock in priority order when
 54075 the lock becomes available. For equal priority threads, write locks shall take precedence over
 54076 read locks. If the Thread Execution Scheduling option is not supported, it is implementation-
 54077 defined whether write locks take precedence over read locks.

54078 Results are undefined if this function is called with an uninitialized read-write lock.

54079 **RETURN VALUE**

54080 If successful, the *pthread_rwlock_unlock()* function shall return zero; otherwise, an error number
 54081 shall be returned to indicate the error.

54082 **ERRORS**

54083 The *pthread_rwlock_unlock()* function shall not return an error code of [EINTR].

54084 **EXAMPLES**

54085 None.

54086 **APPLICATION USAGE**

54087 None.

54088 **RATIONALE**

54089 If an implementation detects that the value specified by the *rwlock* argument to
 54090 *pthread_rwlock_unlock()* does not refer to an initialized read-write lock object, it is recommended
 54091 that the function should fail and report an [EINVAL] error.

54092 If an implementation detects that the value specified by the *rwlock* argument to
 54093 *pthread_rwlock_unlock()* refers to a read-write lock object for which the current thread does not
 54094 hold a lock, it is recommended that the function should fail and report an [EPERM] error.

54095 **FUTURE DIRECTIONS**

54096 None.

SEE ALSO

pthread_rwlock_destroy(), *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*,
pthread_rwlock_timedwrlock(), *pthread_rwlock_trywrlock()*

XBD Section 4.11 (on page 110), <pthread.h>

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The DESCRIPTION is updated as follows:
 - The conditions under which writers have precedence over readers are specified.
 - The concept of read-write lock owner is deleted.
- The SEE ALSO section is updated.

Issue 7

SD5-XSH-ERN-183 is applied.

The *pthread_rwlock_unlock()* function is moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

The [EPERM] error for a read-write lock object for which the current thread does not hold a lock is removed; this condition results in undefined behavior.

pthread_rwlock_wrlock()*System Interfaces*54119 **NAME**

54120 pthread_rwlock_wrlock — lock a read-write lock object for writing

54121 **SYNOPSIS**

54122 #include <pthread.h>

54123 int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);

54124 **DESCRIPTION**54125 Refer to *pthread_rwlock_trywrlock()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

`pthread_rwlockattr_destroy`, `pthread_rwlockattr_init` — destroy and initialize the read-write lock attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```

DESCRIPTION

The `pthread_rwlockattr_destroy()` function shall destroy a read-write lock attributes object. A destroyed `attr` attributes object can be reinitialized using `pthread_rwlockattr_init()`; the results of otherwise referencing the object after it has been destroyed are undefined. An implementation may cause `pthread_rwlockattr_destroy()` to set the object referenced by `attr` to an invalid value.

The `pthread_rwlockattr_init()` function shall initialize a read-write lock attributes object `attr` with the default value for all of the attributes defined by the implementation.

Results are undefined if `pthread_rwlockattr_init()` is called specifying an already initialized `attr` attributes object.

After a read-write lock attributes object has been used to initialize one or more read-write locks, any function affecting the attributes object (including destruction) shall not affect any previously initialized read-write locks.

The behavior is undefined if the value specified by the `attr` argument to `pthread_rwlockattr_destroy()` does not refer to an initialized read-write lock attributes object.

RETURN VALUE

If successful, the `pthread_rwlockattr_destroy()` and `pthread_rwlockattr_init()` functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The `pthread_rwlockattr_init()` function shall fail if:

[ENOMEM] Insufficient memory exists to initialize the read-write lock attributes object.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the `attr` argument to `pthread_rwlockattr_destroy()` does not refer to an initialized read-write lock attributes object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

`pthread_rwlock_destroy()`, `pthread_rwlockattr_getpshared()`

XBD `<pthread.h>`

pthread_rwlockattr_destroy()*System Interfaces***CHANGE HISTORY**

54167
54168 First released in Issue 5.

Issue 6

54169
54170 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 54171 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
54172 now part of the Threads option (previously it was part of the Read-Write Locks option in
54173 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 54174 • The SEE ALSO section is updated.

Issue 7

54175
54176 The *pthread_rwlockattr_destroy()* and *pthread_rwlockattr_init()* functions are moved from the
54177 Threads option to the Base.

54178 The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this
54179 condition results in undefined behavior.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

pthread_rwlockattr_getpshared, pthread_rwlockattr_setpshared — get and set the process-shared attribute of the read-write lock attributes object

SYNOPSIS

```
TSH #include <pthread.h>

int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t
    *restrict attr, int *restrict pshared);
int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_rwlockattr_getpshared()* function shall obtain the value of the *process-shared* attribute from the initialized attributes object referenced by *attr*. The *pthread_rwlockattr_setpshared()* function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute shall be set to *PTHREAD_PROCESS_SHARED* to permit a read-write lock to be operated upon by any thread that has access to the memory where the read-write lock is allocated, even if the read-write lock is allocated in memory that is shared by multiple processes. If the *process-shared* attribute is *PTHREAD_PROCESS_PRIVATE*, the read-write lock shall only be operated upon by threads created within the same process as the thread that initialized the read-write lock; if threads of differing processes attempt to operate on such a read-write lock, the behavior is undefined. The default value of the *process-shared* attribute shall be *PTHREAD_PROCESS_PRIVATE*.

Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are implementation-defined.

The behavior is undefined if the value specified by the *attr* argument to *pthread_rwlockattr_getpshared()* or *pthread_rwlockattr_setpshared()* does not refer to an initialized read-write lock attributes object.

RETURN VALUE

Upon successful completion, the *pthread_rwlockattr_getpshared()* function shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_rwlockattr_setpshared()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlockattr_setpshared()* function may fail if:

[EINVAL] The new value specified for the attribute is outside the range of legal values for that attribute.

These functions shall not return an error code of [EINTR].

pthread_rwlockattr_getpshared()*System Interfaces*54217 **EXAMPLES**

54218 None.

54219 **APPLICATION USAGE**

54220 None.

54221 **RATIONALE**

54222 None.

54223 **FUTURE DIRECTIONS**

54224 None.

54225 **SEE ALSO**54226 *pthread_rwlock_destroy()*, *pthread_rwlockattr_destroy()*

54227 XBD <pthread.h>

54228 **CHANGE HISTORY**

54229 First released in Issue 5.

54230 **Issue 6**

54231 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 54232 • The margin code in the SYNOPSIS is changed to THR TSH to indicate that the
- 54233 functionality is now part of the Threads option (previously it was part of the Read-Write
- 54234 Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 54235 • The DESCRIPTION notes that additional attributes are implementation-defined.
- 54236 • The SEE ALSO section is updated.

54237 The **restrict** keyword is added to the *pthread_rwlockattr_getpshared()* prototype for alignment

54238 with the ISO/IEC 9899:1999 standard.

54239 **Issue 7**

54240 The *pthread_rwlockattr_getpshared()* and *pthread_rwlockattr_setpshared()* functions are moved from

54241 the Threads option.

54242 The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this

54243 condition results in undefined behavior.

*System Interfaces***pthread_rwlockattr_init()**54244 **NAME**

54245 pthread_rwlockattr_init — initialize the read-write lock attributes object

54246 **SYNOPSIS**

54247 #include <pthread.h>

54248 int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);

54249 **DESCRIPTION**54250 Refer to *pthread_rwlockattr_destroy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_rwlockattr_setpshared()*System Interfaces*54251 **NAME**

54252 pthread_rwlockattr_setpshared — set the process-shared attribute of the read-write lock
54253 attributes object

54254 **SYNOPSIS**

```
54255 TSH #include <pthread.h>  
54256      int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,  
54257      int pshared);
```

54258 **DESCRIPTION**

54259 Refer to *pthread_rwlockattr_getpshared()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

54260 NAME

54261 pthread_self — get the calling thread ID

54262 SYNOPSIS

54263 #include <pthread.h>

54264 pthread_t pthread_self(void);

54265 DESCRIPTION

54266 The *pthread_self()* function shall return the thread ID of the calling thread.

54267 RETURN VALUE

54268 The *pthread_self()* function shall always be successful and no return value is reserved to indicate
54269 an error.

54270 ERRORS

54271 No errors are defined.

54272 EXAMPLES

54273 None.

54274 APPLICATION USAGE

54275 None.

54276 RATIONALE

54277 The *pthread_self()* function provides a capability similar to the *getpid()* function for processes
54278 and the rationale is the same: the creation call does not provide the thread ID to the created
54279 thread.

54280 FUTURE DIRECTIONS

54281 None.

54282 SEE ALSO

54283 *pthread_create()*, *pthread_equal()*

54284 XBD <pthread.h>

54285 CHANGE HISTORY

54286 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

54287 Issue 6

54288 The *pthread_self()* function is marked as part of the Threads option.

54289 Issue 7

54290 Austin Group Interpretation 1003.1-2001 #063 is applied, updating the RETURN VALUE section.

54291 The *pthread_self()* function is moved from the Threads option to the Base.

pthread_setcancelstate()

System Interfaces

54292 **NAME**

54293 pthread_setcancelstate, pthread_setcanceltype, pthread_testcancel — set cancelability state

54294 **SYNOPSIS**

```
54295 #include <pthread.h>
54296 int pthread_setcancelstate(int state, int *oldstate);
54297 int pthread_setcanceltype(int type, int *oldtype);
54298 void pthread_testcancel(void);
```

54299 **DESCRIPTION**

54300 The *pthread_setcancelstate()* function shall atomically both set the calling thread's cancelability
 54301 state to the indicated *state* and return the previous cancelability state at the location referenced
 54302 by *oldstate*. Legal values for *state* are PTHREAD_CANCEL_ENABLE and
 54303 PTHREAD_CANCEL_DISABLE.

54304 The *pthread_setcanceltype()* function shall atomically both set the calling thread's cancelability
 54305 type to the indicated *type* and return the previous cancelability type at the location referenced by
 54306 *oldtype*. Legal values for *type* are PTHREAD_CANCEL_DEFERRED and
 54307 PTHREAD_CANCEL_ASYNCHRONOUS.

54308 The cancelability state and type of any newly created threads, including the thread in which
 54309 *main()* was first invoked, shall be PTHREAD_CANCEL_ENABLE and
 54310 PTHREAD_CANCEL_DEFERRED respectively.

54311 The *pthread_testcancel()* function shall create a cancellation point in the calling thread. The
 54312 *pthread_testcancel()* function shall have no effect if cancelability is disabled.

54313 **RETURN VALUE**

54314 If successful, the *pthread_setcancelstate()* and *pthread_setcanceltype()* functions shall return zero;
 54315 otherwise, an error number shall be returned to indicate the error.

54316 **ERRORS**

54317 The *pthread_setcancelstate()* function may fail if:

54318 [EINVAL] The specified state is not PTHREAD_CANCEL_ENABLE or
 54319 PTHREAD_CANCEL_DISABLE.

54320 The *pthread_setcanceltype()* function may fail if:

54321 [EINVAL] The specified type is not PTHREAD_CANCEL_DEFERRED or
 54322 PTHREAD_CANCEL_ASYNCHRONOUS.

54323 These functions shall not return an error code of [EINTR].

54324 **EXAMPLES**

54325 None.

54326 **APPLICATION USAGE**

54327 None.

54328 **RATIONALE**

54329 The *pthread_setcancelstate()* and *pthread_setcanceltype()* functions control the points at which a
 54330 thread may be asynchronously canceled. For cancellation control to be usable in modular
 54331 fashion, some rules need to be followed.

54332 An object can be considered to be a generalization of a procedure. It is a set of procedures and
 54333 global variables written as a unit and called by clients not known by the object. Objects may
 54334 depend on other objects.

54335 First, cancelability should only be disabled on entry to an object, never explicitly enabled. On

54336 exit from an object, the cancelability state should always be restored to its value on entry to the
 54337 object.

54338 This follows from a modularity argument: if the client of an object (or the client of an object that
 54339 uses that object) has disabled cancelability, it is because the client does not want to be concerned
 54340 about cleaning up if the thread is canceled while executing some sequence of actions. If an object
 54341 is called in such a state and it enables cancelability and a cancellation request is pending for that
 54342 thread, then the thread is canceled, contrary to the wish of the client that disabled.

54343 Second, the cancelability type may be explicitly set to either *deferred* or *asynchronous* upon entry
 54344 to an object. But as with the cancelability state, on exit from an object the cancelability type
 54345 should always be restored to its value on entry to the object.

54346 Finally, only functions that are cancel-safe may be called from a thread that is asynchronously
 54347 cancelable.

54348 **FUTURE DIRECTIONS**

54349 None.

54350 **SEE ALSO**

54351 [*pthread_cancel\(\)*](#)

54352 XBD [*<pthread.h>*](#)

54353 **CHANGE HISTORY**

54354 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

54355 **Issue 6**

54356 The *pthread_setcancelstate()*, *pthread_setcanceltype()*, and *pthread_testcancel()* functions are marked
 54357 as part of the Threads option.

54358 **Issue 7**

54359 The *pthread_setcancelstate()*, *pthread_setcanceltype()*, and *pthread_testcancel()* functions are moved
 54360 from the Threads option to the Base.

pthread_setconcurrency()*System Interfaces*54361 **NAME**

54362 pthread_setconcurrency — set the level of concurrency

54363 **SYNOPSIS**54364 OB XSI `#include <pthread.h>`54365 `int pthread_setconcurrency(int new_level);`54366 **DESCRIPTION**54367 Refer to *pthread_getconcurrency()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread_setschedparam()**54368 **NAME**

54369 pthread_setschedparam — dynamic thread scheduling parameters access (REALTIME
54370 THREADS)

54371 **SYNOPSIS**

```
54372 TPS #include <pthread.h>  
54373  
54373 int pthread_setschedparam(pthread_t thread, int policy,  
54374 const struct sched_param *param);
```

54375 **DESCRIPTION**

54376 Refer to *pthread_getschedparam()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_setschedprio()

System Interfaces

54377 **NAME**

54378 pthread_setschedprio — dynamic thread scheduling parameters access (**REALTIME**
 54379 **THREADS**)

54380 **SYNOPSIS**

```
54381 TPS #include <pthread.h>
54382 int pthread_setschedprio(pthread_t thread, int prio);
```

54383 **DESCRIPTION**

54384 The *pthread_setschedprio()* function shall set the scheduling priority for the thread whose thread
 54385 ID is given by *thread* to the value given by *prio*. See [Scheduling Policies](#) (on page 501) for a
 54386 description on how this function call affects the ordering of the thread in the thread list for its
 54387 new priority.

54388 If the *pthread_setschedprio()* function fails, the scheduling priority of the target thread shall not be
 54389 changed.

54390 **RETURN VALUE**

54391 If successful, the *pthread_setschedprio()* function shall return zero, otherwise, an error number
 54392 shall be returned to indicate the error.

54393 **ERRORS**

54394 The *pthread_setschedprio()* function may fail if:

- 54395 [EINVAL] The value of *prio* is invalid for the scheduling policy of the specified thread.
- 54396 [ENOTSUP] An attempt was made to set the priority to an unsupported value.
- 54397 [EPERM] The caller does not have appropriate privileges to set the scheduling priority
 54398 of the specified thread.

54399 The *pthread_setschedprio()* function shall not return an error code of [EINTR].

54400 **EXAMPLES**

54401 None.

54402 **APPLICATION USAGE**

54403 None.

54404 **RATIONALE**

54405 The *pthread_setschedprio()* function provides a way for an application to temporarily raise its
 54406 priority and then lower it again, without having the undesired side-effect of yielding to other
 54407 threads of the same priority. This is necessary if the application is to implement its own
 54408 strategies for bounding priority inversion, such as priority inheritance or priority ceilings. This
 54409 capability is especially important if the implementation does not support the Thread Priority
 54410 Protection or Thread Priority Inheritance options, but even if those options are supported it is
 54411 needed if the application is to bound priority inheritance for other resources, such as
 54412 semaphores.

54413 The standard developers considered that while it might be preferable conceptually to solve this
 54414 problem by modifying the specification of *pthread_setschedparam()*, it was too late to make such a
 54415 change, as there may be implementations that would need to be changed. Therefore, this new
 54416 function was introduced.

54417 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
 54418 that the function should fail and report an [ESRCH] error.

54419 **FUTURE DIRECTIONS**

54420 None.

54421 **SEE ALSO**54422 [Scheduling Policies](#) (on page 501), [pthread_getschedparam\(\)](#)54423 XBD [<pthread.h>](#)54424 **CHANGE HISTORY**

54425 First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96.

54426 **Issue 7**54427 The *pthread_setschedprio()* function is moved from the Threads option.54428 Austin Group Interpretation 1003.1-2001 #069 is applied, updating the [E_{PERM}] error.54429 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [E_{SRCH}] error condition.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pthread_setspecific()*System Interfaces*54430 **NAME**

54431 pthread_setspecific — thread-specific data management

54432 **SYNOPSIS**

54433 #include <pthread.h>

54434 int pthread_setspecific(pthread_key_t *key*, const void **value*);54435 **DESCRIPTION**54436 Refer to *pthread_getspecific()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

pthread_sigmask, sigprocmask — examine and change blocked signals

SYNOPSIS

```
#include <signal.h>

int pthread_sigmask(int how, const sigset_t *restrict set,
                    sigset_t *restrict oset);
int sigprocmask(int how, const sigset_t *restrict set,
                sigset_t *restrict oset);
```

DESCRIPTION

The *pthread_sigmask()* function shall examine or change (or both) the calling thread's signal mask, regardless of the number of threads in the process. The function shall be equivalent to *sigprocmask()*, without the restriction that the call be made in a single-threaded process.

In a single-threaded process, the *sigprocmask()* function shall examine or change (or both) the signal mask of the calling thread.

If the argument *set* is not a null pointer, it points to a set of signals to be used to change the currently blocked set.

The argument *how* indicates the way in which the set is changed, and the application shall ensure it consists of one of the following values:

SIG_BLOCK The resulting set shall be the union of the current set and the signal set pointed to by *set*.

SIG_SETMASK The resulting set shall be the signal set pointed to by *set*.

SIG_UNBLOCK The resulting set shall be the intersection of the current set and the complement of the signal set pointed to by *set*.

If the argument *oset* is not a null pointer, the previous mask shall be stored in the location pointed to by *oset*. If *set* is a null pointer, the value of the argument *how* is not significant and the thread's signal mask shall be unchanged; thus the call can be used to enquire about currently blocked signals.

If there are any pending unblocked signals after the call to *sigprocmask()*, at least one of those signals shall be delivered before the call to *sigprocmask()* returns.

It is not possible to block those signals which cannot be ignored. This shall be enforced by the system without causing an error to be indicated.

If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked, the result is undefined, unless the signal was generated by the *kill()* function, the *sigqueue()* function, or the *raise()* function.

If *sigprocmask()* fails, the thread's signal mask shall not be changed.

The use of the *sigprocmask()* function is unspecified in a multi-threaded process.

RETURN VALUE

Upon successful completion *pthread_sigmask()* shall return 0; otherwise, it shall return the corresponding error number.

Upon successful completion, *sigprocmask()* shall return 0; otherwise, -1 shall be returned, *errno* shall be set to indicate the error, and the signal mask of the process shall be unchanged.

pthread_sigmask()*System Interfaces***54478 ERRORS**

- 54479 The *pthread_sigmask()* and *sigprocmask()* functions shall fail if:
- 54480 [EINVAL] The value of the *how* argument is not equal to one of the defined values.
- 54481 The *pthread_sigmask()* function shall not return an error code of [EINTR].

54482 EXAMPLES**54483 Signaling in a Multi-Threaded Process**

54484 This example shows the use of *pthread_sigmask()* in order to deal with signals in a multi-
 54485 threaded process. It provides a fairly general framework that could be easily adapted/extended.

```

54486 #include <stdio.h>
54487 #include <stdlib.h>
54488 #include <pthread.h>
54489 #include <signal.h>
54490 #include <string.h>
54491 #include <errno.h>
54492 ...

54493 static sigset_t  signal_mask; /* signals to block          */
54494 int main (int argc, char *argv[])
54495 {
54496     pthread_t  sig_thr_id;      /* signal handler thread ID */
54497     int        rc;              /* return code               */

54498     sigemptyset (&signal_mask);
54499     sigaddset (&signal_mask, SIGINT);
54500     sigaddset (&signal_mask, SIGTERM);
54501     rc = pthread_sigmask (SIG_BLOCK, &signal_mask, NULL);
54502     if (rc != 0) {
54503         /* handle error */
54504         ...
54505     }
54506     /* any newly created threads inherit the signal mask */

54507     rc = pthread_create (&sig_thr_id, NULL, signal_thread, NULL);
54508     if (rc != 0) {
54509         /* handle error */
54510         ...
54511     }

54512     /* APPLICATION CODE */
54513     ...
54514 }

54515 void *signal_thread (void *arg)
54516 {
54517     int        sig_caught;      /* signal caught             */
54518     int        rc;              /* returned code             */

54519     rc = sigwait (&signal_mask, &sig_caught);
54520     if (rc != 0) {
54521         /* handle error */
54522     }

```

```

54523         switch (sig_caught)
54524         {
54525             case SIGINT:      /* process SIGINT */
54526                 ...
54527                 break;
54528             case SIGTERM:     /* process SIGTERM */
54529                 ...
54530                 break;
54531             default:          /* should normally not happen */
54532                 fprintf (stderr, "\nUnexpected signal %d\n", sig_caught);
54533                 break;
54534         }
54535     }

```

APPLICATION USAGE

None.

RATIONALE

When a thread's signal mask is changed in a signal-catching function that is installed by *sigaction()*, the restoration of the signal mask on return from the signal-catching function overrides that change (see *sigaction()*). If the signal-catching function was installed with *signal()*, it is unspecified whether this occurs.

See *kill()* for a discussion of the requirement on delivery of signals.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *kill()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *sigpending()*, *sigqueue()*, *sigsuspend()*.

XBD <signal.h>

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

The *pthread_sigmask()* function is added for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_sigmask()* function is marked as part of the Threads option.

The SYNOPSIS for *sigprocmask()* is marked as a CX extension to note that the presence of this function in the <signal.h> header is an extension to the ISO C standard.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The DESCRIPTION is updated to explicitly state the functions which may generate the signal.

The normative text is updated to avoid use of the term “must” for application requirements.

The **restrict** keyword is added to the *pthread_sigmask()* and *sigprocmask()* prototypes for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/105 is applied, updating “process’ signal mask” to “thread’s signal mask” in the DESCRIPTION and RATIONALE sections.

pthread_sigmask()*System Interfaces*

54567	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/106 is applied, adding the example to the
54568	EXAMPLES section.
54569	Issue 7
54570	The <i>pthread_sigmask()</i> function is moved from the Threads option to the Base.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

pthread_spin_destroy, pthread_spin_init — destroy or initialize a spin lock object

SYNOPSIS

```
#include <pthread.h>

int pthread_spin_destroy(pthread_spinlock_t *lock);
int pthread_spin_init(pthread_spinlock_t *lock, int pshared);
```

DESCRIPTION

The *pthread_spin_destroy()* function shall destroy the spin lock referenced by *lock* and release any resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is reinitialized by another call to *pthread_spin_init()*. The results are undefined if *pthread_spin_destroy()* is called when a thread holds the lock, or if this function is called with an uninitialized thread spin lock.

The *pthread_spin_init()* function shall allocate any resources required to use the spin lock referenced by *lock* and initialize the lock to an unlocked state.

TSH

If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is *PTHREAD_PROCESS_SHARED*, the implementation shall permit the spin lock to be operated upon by any thread that has access to the memory where the spin lock is allocated, even if it is allocated in memory that is shared by multiple processes.

If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is *PTHREAD_PROCESS_PRIVATE*, or if the option is not supported, the spin lock shall only be operated upon by threads created within the same process as the thread that initialized the spin lock. If threads of differing processes attempt to operate on such a spin lock, the behavior is undefined.

The results are undefined if *pthread_spin_init()* is called specifying an already initialized spin lock. The results are undefined if a spin lock is used without first being initialized.

If the *pthread_spin_init()* function fails, the lock is not initialized and the contents of *lock* are undefined.

Only the object referenced by *lock* may be used for performing synchronization.

The result of referring to copies of that object in calls to *pthread_spin_destroy()*, *pthread_spin_lock()*, *pthread_spin_trylock()*, or *pthread_spin_unlock()* is undefined.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_spin_init()* function shall fail if:

[EAGAIN] The system lacks the necessary resources to initialize another spin lock.

[ENOMEM] Insufficient memory exists to initialize the lock.

These functions shall not return an error code of [EINTR].

pthread_spin_destroy()

System Interfaces

54609 **EXAMPLES**

54610 None.

54611 **APPLICATION USAGE**

54612 None.

54613 **RATIONALE**

54614 If an implementation detects that the value specified by the *lock* argument to
 54615 *pthread_spin_destroy()* does not refer to an initialized spin lock object, it is recommended that the
 54616 function should fail and report an [EINVAL] error.

54617 If an implementation detects that the value specified by the *lock* argument to
 54618 *pthread_spin_destroy()* or *pthread_spin_init()* refers to a locked spin lock object, or detects that the
 54619 value specified by the *lock* argument to *pthread_spin_init()* refers to an already initialized spin
 54620 lock object, it is recommended that the function should fail and report an [EBUSY] error.

54621 **FUTURE DIRECTIONS**

54622 None.

54623 **SEE ALSO**54624 *pthread_spin_lock()*, *pthread_spin_unlock()*54625 XBD **<pthread.h>**54626 **CHANGE HISTORY**

54627 First released in Issue 6. Derived from IEEE Std 1003.1-2000.

54628 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.54629 **Issue 7**

54630 The *pthread_spin_destroy()* and *pthread_spin_init()* functions are moved from the Spin Locks
 54631 option to the Base.

54632 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in
 54633 undefined behavior.

54634 The [EBUSY] error for a locked spin lock object or an already initialized spin lock object is
 54635 removed; this condition results in undefined behavior.

NAME

pthread_spin_lock, pthread_spin_trylock — lock a spin lock object

SYNOPSIS

```
#include <pthread.h>

int pthread_spin_lock(pthread_spinlock_t *lock);
int pthread_spin_trylock(pthread_spinlock_t *lock);
```

DESCRIPTION

The *pthread_spin_lock()* function shall lock the spin lock referenced by *lock*. The calling thread shall acquire the lock if it is not held by another thread. Otherwise, the thread shall spin (that is, shall not return from the *pthread_spin_lock()* call) until the lock becomes available. The results are undefined if the calling thread holds the lock at the time the call is made. The *pthread_spin_trylock()* function shall lock the spin lock referenced by *lock* if it is not held by any thread. Otherwise, the function shall fail.

The results are undefined if any of these functions is called with an uninitialized spin lock.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_spin_lock()* function may fail if:

[EDEADLK] A deadlock condition was detected.

The *pthread_spin_trylock()* function shall fail if:

[EBUSY] A thread currently holds the lock.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using this function may be subject to priority inversion, as discussed in XBD Section 3.285 (on page 79).

RATIONALE

If an implementation detects that the value specified by the *lock* argument to *pthread_spin_lock()* or *pthread_spin_trylock()* does not refer to an initialized spin lock object, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *lock* argument to *pthread_spin_lock()* refers to a spin lock object for which the calling thread already holds the lock, it is recommended that the function should fail and report an [EDEADLK] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_spin_destroy(), *pthread_spin_unlock()*

XBD Section 3.285 (on page 79), Section 4.11 (on page 110), **<pthread.h>**

pthread_spin_lock()*System Interfaces*54676 **CHANGE HISTORY**

54677 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

54678 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.54679 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/107 is applied, updating the ERRORS
54680 section so that the [EDEADLK] error includes detection of a deadlock condition.54681 **Issue 7**54682 The *pthread_spin_lock()* and *pthread_spin_trylock()* functions are moved from the Spin Locks
54683 option to the Base.54684 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in
54685 undefined behavior.54686 The [EDEADLK] error for a spin lock object for which the calling thread already holds the lock is
54687 removed; this condition results in undefined behavior.

NAME

pthread_spin_unlock — unlock a spin lock object

SYNOPSIS

```
#include <pthread.h>

int pthread_spin_unlock(pthread_spinlock_t *lock);
```

DESCRIPTION

The *pthread_spin_unlock()* function shall release the spin lock referenced by *lock* which was locked via the *pthread_spin_lock()* or *pthread_spin_trylock()* functions.

The results are undefined if the lock is not held by the calling thread.

If there are threads spinning on the lock when *pthread_spin_unlock()* is called, the lock becomes available and an unspecified spinning thread shall acquire the lock.

The results are undefined if this function is called with an uninitialized thread spin lock.

RETURN VALUE

Upon successful completion, the *pthread_spin_unlock()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

This function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *lock* argument to *pthread_spin_unlock()* does not refer to an initialized spin lock object, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *lock* argument to *pthread_spin_unlock()* refers to a spin lock object for which the current thread does not hold the lock, it is recommended that the function should fail and report an [EPERM] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_spin_destroy(), *pthread_spin_lock()*

XBD Section 4.11 (on page 110), *<pthread.h>*

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

In the SYNOPSIS, the inclusion of *<sys/types.h>* is no longer required.

Issue 7

The *pthread_spin_unlock()* function is moved from the Spin Locks option to the Base.

The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in undefined behavior.

pthread_spin_unlock()*System Interfaces*54728
54729

The [EPERM] error for a spin lock object for which the current thread does not hold the lock is removed; this condition results in undefined behavior.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pthread_testcancel()**54730 **NAME**

54731 pthread_testcancel — set cancelability state

54732 **SYNOPSIS**

54733 #include <pthread.h>

54734 void pthread_testcancel(void);

54735 **DESCRIPTION**54736 Refer to *pthread_setcancelstate()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

ptsname()

System Interfaces

54737 **NAME**

54738 ptsname — get name of the slave pseudo-terminal device

54739 **SYNOPSIS**

```
54740 XSI      #include <stdlib.h>
54741      char *ptsname(int fildes);
```

54742 **DESCRIPTION**

54743 The *ptsname()* function shall return the name of the slave pseudo-terminal device associated
 54744 with a master pseudo-terminal device. The *fildes* argument is a file descriptor that refers to the
 54745 master device. The *ptsname()* function shall return a pointer to a string containing the pathname
 54746 of the corresponding slave device.

54747 The *ptsname()* function need not be thread-safe.

54748 **RETURN VALUE**

54749 Upon successful completion, *ptsname()* shall return a pointer to a string which is the name of the
 54750 pseudo-terminal slave device. Upon failure, *ptsname()* shall return a null pointer. This could
 54751 occur if *fildes* is an invalid file descriptor or if the slave device name does not exist in the file
 54752 system.

54753 **ERRORS**

54754 No errors are defined.

54755 **EXAMPLES**

54756 None.

54757 **APPLICATION USAGE**

54758 The value returned may point to a static data area that is overwritten by each call to *ptsname()*.

54759 **RATIONALE**

54760 None.

54761 **FUTURE DIRECTIONS**

54762 None.

54763 **SEE ALSO**

54764 *grantpt()*, *open()*, *ttyname()*, *unlockpt()*

54765 XBD <stdlib.h>

54766 **CHANGE HISTORY**

54767 First released in Issue 4, Version 2.

54768 **Issue 5**

54769 Moved from X/OPEN UNIX extension to BASE.

54770 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

54771 **Issue 7**

54772 Austin Group Interpretation 1003.1-2001 #156 is applied.

54773 **NAME**

54774 putc — put a byte on a stream

54775 **SYNOPSIS**

54776 #include <stdio.h>

54777 int putc(int *c*, FILE **stream*);54778 **DESCRIPTION**

54779 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 54780 conflict between the requirements described here and the ISO C standard is unintentional. This
 54781 volume of POSIX.1-2008 defers to the ISO C standard.

54782 The *putc()* function shall be equivalent to *fputc()*, except that if it is implemented as a macro it
 54783 may evaluate *stream* more than once, so the argument should never be an expression with side-
 54784 effects.

54785 **RETURN VALUE**54786 Refer to *fputc()*.54787 **ERRORS**54788 Refer to *fputc()*.54789 **EXAMPLES**

54790 None.

54791 **APPLICATION USAGE**

54792 Since it may be implemented as a macro, *putc()* may treat a *stream* argument with side-effects
 54793 incorrectly. In particular, *putc(c,*f++)* does not necessarily work correctly. Therefore, use of this
 54794 function is not recommended in such situations; *fputc()* should be used instead.

54795 **RATIONALE**

54796 None.

54797 **FUTURE DIRECTIONS**

54798 None.

54799 **SEE ALSO**54800 *fputc()*

54801 XBD <stdio.h>

54802 **CHANGE HISTORY**

54803 First released in Issue 1. Derived from Issue 1 of the SVID.

putc_unlocked()*System Interfaces*54804 **NAME**

54805 putc_unlocked — stdio with explicit client locking

54806 **SYNOPSIS**

```
54807 CX     #include <stdio.h>  
54808     int putc_unlocked(int c, FILE *stream);
```

54809 **DESCRIPTION**54810 Refer to *getc_unlocked()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

putchar()54811 **NAME**54812 **putchar** — put a byte on a stdout stream54813 **SYNOPSIS**54814 `#include <stdio.h>`54815 `int putchar(int c);`54816 **DESCRIPTION**

54817 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 54818 conflict between the requirements described here and the ISO C standard is unintentional. This
 54819 volume of POSIX.1-2008 defers to the ISO C standard.

54820 The function call *putchar(c)* shall be equivalent to *putc(c,stdout)*.54821 **RETURN VALUE**54822 Refer to *fputc()*.54823 **ERRORS**54824 Refer to *fputc()*.54825 **EXAMPLES**

54826 None.

54827 **APPLICATION USAGE**

54828 None.

54829 **RATIONALE**

54830 None.

54831 **FUTURE DIRECTIONS**

54832 None.

54833 **SEE ALSO**54834 *putc()*54835 XBD `<stdio.h>`54836 **CHANGE HISTORY**

54837 First released in Issue 1. Derived from Issue 1 of the SVID.

putchar_unlocked()*System Interfaces*54838 **NAME**

54839 putchar_unlocked — stdio with explicit client locking

54840 **SYNOPSIS**54841 CX `#include <stdio.h>`54842 `int putchar_unlocked(int c);`54843 **DESCRIPTION**54844 Refer to *getc_unlocked()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

putenv — change or add a value to an environment

SYNOPSIS

```
#include <stdlib.h>

int putenv(char *string);
```

DESCRIPTION

The *putenv()* function shall use the *string* argument to set environment variable values. The *string* argument should point to a string of the form "*name=value*". The *putenv()* function shall make the value of the environment variable *name* equal to *value* by altering an existing variable or creating a new one. In either case, the string pointed to by *string* shall become part of the environment, so altering the string shall change the environment. The space used by *string* is no longer used once a new string which defines *name* is passed to *putenv()*.

The *putenv()* function need not be thread-safe.

RETURN VALUE

Upon successful completion, *putenv()* shall return 0; otherwise, it shall return a non-zero value and set *errno* to indicate the error.

ERRORS

The *putenv()* function may fail if:

[ENOMEM] Insufficient memory was available.

EXAMPLES**Changing the Value of an Environment Variable**

The following example changes the value of the *HOME* environment variable to the value */usr/home*.

```
#include <stdlib.h>
...
static char *var = "HOME=/usr/home";
int ret;

ret = putenv(var);
```

APPLICATION USAGE

The *putenv()* function manipulates the environment pointed to by *environ*, and can be used in conjunction with *getenv()*.

See *exec()* for restrictions on changing the environment in multi-threaded applications.

This routine may use *malloc()* to enlarge the environment.

A potential error is to call *putenv()* with an automatic variable as the argument, then return from the calling function while *string* is still part of the environment.

The *setenv()* function is preferred over this function.

RATIONALE

The standard developers noted that *putenv()* is the only function available to add to the environment without permitting memory leaks.

putenv()

System Interfaces

54884 **FUTURE DIRECTIONS**

54885 None.

54886 **SEE ALSO**54887 *exec*, *getenv()*, *malloc()*, *setenv()*54888 XBD *<stdlib.h>*54889 **CHANGE HISTORY**

54890 First released in Issue 1. Derived from Issue 1 of the SVID.

54891 **Issue 5**54892 The type of the argument to this function is changed from **const char *** to **char ***. This was
54893 indicated as a FUTURE DIRECTION in previous issues.

54894 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

54895 **Issue 6**54896 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/48 is applied, clarifying wording in the
54897 DESCRIPTION and adding a new paragraph into APPLICATION USAGE referring readers to
54898 *exec*.54899 **Issue 7**

54900 Austin Group Interpretation 1003.1-2001 #156 is applied.

NAME

putmsg, putpmsg — send a message on a STREAM (STREAMS)

SYNOPSIS

```
#include <stropts.h>

int putmsg(int fildes, const struct strbuf *ctlptr,
           const struct strbuf *dataptr, int flags);
int putpmsg(int fildes, const struct strbuf *ctlptr,
            const struct strbuf *dataptr, int band, int flags);
```

DESCRIPTION

The *putmsg()* function shall create a message from a process buffer(s) and send the message to a STREAMS file. The message may contain either a data part, a control part, or both. The data and control parts are distinguished by placement in separate buffers, as described below. The semantics of each part are defined by the STREAMS module that receives the message.

The *putpmsg()* function is equivalent to *putmsg()*, except that the process can send messages in different priority bands. Except where noted, all requirements on *putmsg()* also pertain to *putpmsg()*.

The *fildes* argument specifies a file descriptor referencing an open STREAM. The *ctlptr* and *dataptr* arguments each point to a **strbuf** structure.

The *ctlptr* argument points to the structure describing the control part, if any, to be included in the message. The *buf* member in the **strbuf** structure points to the buffer where the control information resides, and the *len* member indicates the number of bytes to be sent. The *maxlen* member is not used by *putmsg()*. In a similar manner, the argument *dataptr* specifies the data, if any, to be included in the message. The *flags* argument indicates what type of message should be sent and is described further below.

To send the data part of a message, the application shall ensure that *dataptr* is not a null pointer and the *len* member of *dataptr* is 0 or greater. To send the control part of a message, the application shall ensure that the corresponding values are set for *ctlptr*. No data (control) part shall be sent if either *dataptr*(*ctlptr*) is a null pointer or the *len* member of *dataptr*(*ctlptr*) is set to -1.

For *putmsg()*, if a control part is specified and *flags* is set to RS_HIPRI, a high priority message shall be sent. If no control part is specified, and *flags* is set to RS_HIPRI, *putmsg()* shall fail and set *errno* to [EINVAL]. If *flags* is set to 0, a normal message (priority band equal to 0) shall be sent. If a control part and data part are not specified and *flags* is set to 0, no message shall be sent and 0 shall be returned.

For *putpmsg()*, the flags are different. The *flags* argument is a bitmask with the following mutually-exclusive flags defined: MSG_HIPRI and MSG_BAND. If *flags* is set to 0, *putpmsg()* shall fail and set *errno* to [EINVAL]. If a control part is specified and *flags* is set to MSG_HIPRI and *band* is set to 0, a high-priority message shall be sent. If *flags* is set to MSG_HIPRI and either no control part is specified or *band* is set to a non-zero value, *putpmsg()* shall fail and set *errno* to [EINVAL]. If *flags* is set to MSG_BAND, then a message shall be sent in the priority band specified by *band*. If a control part and data part are not specified and *flags* is set to MSG_BAND, no message shall be sent and 0 shall be returned.

putmsg()*System Interfaces*

54943 The *putmsg()* function shall block if the STREAM write queue is full due to internal flow control
54944 conditions, with the following exceptions:

- 54945 • For high-priority messages, *putmsg()* shall not block on this condition and continues
54946 processing the message.
- 54947 • For other messages, *putmsg()* shall not block but shall fail when the write queue is full and
54948 *O_NONBLOCK* is set.

54949 The *putmsg()* function shall also block, unless prevented by lack of internal resources, while
54950 waiting for the availability of message blocks in the STREAM, regardless of priority or whether
54951 *O_NONBLOCK* has been specified. No partial message shall be sent.

RETURN VALUE

54952 Upon successful completion, *putmsg()* and *putpmsg()* shall return 0; otherwise, they shall return
54953 -1 and set *errno* to indicate the error.

ERRORS

54955 The *putmsg()* and *putpmsg()* functions shall fail if:

54957 54958 54959	[EAGAIN]	A non-priority message was specified, the <i>O_NONBLOCK</i> flag is set, and the STREAM write queue is full due to internal flow control conditions; or buffers could not be allocated for the message that was to be created.
54960	[EBADF]	<i>fildev</i> is not a valid file descriptor open for writing.
54961	[EINTR]	A signal was caught during <i>putmsg()</i> .
54962 54963 54964 54965 54966	[EINVAL]	An undefined value is specified in <i>flags</i> , or <i>flags</i> is set to <i>RS_HIPRI</i> or <i>MSG_HIPRI</i> and no control part is supplied, or the STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer, or <i>flags</i> is set to <i>MSG_HIPRI</i> and <i>band</i> is non-zero (for <i>putpmsg()</i> only).
54967 54968	[ENOSR]	Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.
54969	[ENOSTR]	A STREAM is not associated with <i>fildev</i> .
54970	[ENXIO]	A hangup condition was generated downstream for the specified STREAM.
54971 54972	[EPIPE] or [EIO]	The <i>fildev</i> argument refers to a STREAMS-based pipe and the other end of the pipe is closed. A SIGPIPE signal is generated for the calling thread.
54973 54974 54975 54976 54977 54978	[ERANGE]	The size of the data part of the message does not fall within the range specified by the maximum and minimum packet sizes of the topmost STREAM module. This value is also returned if the control part of the message is larger than the maximum configured size of the control part of a message, or if the data part of a message is larger than the maximum configured size of the data part of a message.

54979 In addition, *putmsg()* and *putpmsg()* shall fail if the STREAM head had processed an
54980 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of
54981 *putmsg()* or *putpmsg()*, but reflects the prior error.

EXAMPLES

Sending a High-Priority Message

The value of *fd* is assumed to refer to an open STREAMS file. This call to *putmsg()* does the following:

1. Creates a high-priority message with a control part and a data part, using the buffers pointed to by *ctrlbuf* and *databuf*, respectively.
2. Sends the message to the STREAMS file identified by *fd*.

```
#include <stropts.h>
#include <string.h>
...
int fd;
char *ctrlbuf = "This is the control part";
char *databuf = "This is the data part";
struct strbuf ctrl;
struct strbuf data;
int ret;

ctrl.buf = ctrlbuf;
ctrl.len = strlen(ctrlbuf);

data.buf = databuf;
data.len = strlen(databuf);

ret = putmsg(fd, &ctrl, &data, MSG_HIPRI);
```

Using putpmsg()

This example has the same effect as the previous example. In this example, however, the *putpmsg()* function creates and sends the message to the STREAMS file.

```
#include <stropts.h>
#include <string.h>
...
int fd;
char *ctrlbuf = "This is the control part";
char *databuf = "This is the data part";
struct strbuf ctrl;
struct strbuf data;
int ret;

ctrl.buf = ctrlbuf;
ctrl.len = strlen(ctrlbuf);

data.buf = databuf;
data.len = strlen(databuf);

ret = putpmsg(fd, &ctrl, &data, 0, MSG_HIPRI);
```

APPLICATION USAGE

None.

putmsg()*System Interfaces*55022 **RATIONALE**

55023 None.

55024 **FUTURE DIRECTIONS**55025 The *putmsg()* and *putpmsg()* functions may be removed in a future version.55026 **SEE ALSO**55027 [Section 2.6](#) (on page 494), *getmsg()*, *poll()*, *read()*, *write()*55028 XBD [<stropts.h>](#)55029 **CHANGE HISTORY**

55030 First released in Issue 4, Version 2.

55031 **Issue 5**

55032 Moved from X/OPEN UNIX extension to BASE.

55033 The following text is removed from the DESCRIPTION: “The STREAM head guarantees that the
 55034 control part of a message generated by *putmsg()* is at least 64 bytes in length”.

55035 **Issue 6**

55036 This function is marked as part of the XSI STREAMS Option Group.

55037 The normative text is updated to avoid use of the term “must” for application requirements.

55038 **Issue 7**55039 The *putmsg()* and *putpmsg()* functions are marked obsolescent.

System Interfaces

puts()55040 **NAME**

55041 puts — put a string on standard output

55042 **SYNOPSIS**

55043 #include <stdio.h>

55044 int puts(const char *s);

55045 **DESCRIPTION**

55046 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 55047 conflict between the requirements described here and the ISO C standard is unintentional. This
 55048 volume of POSIX.1-2008 defers to the ISO C standard.

55049 The *puts()* function shall write the string pointed to by *s*, followed by a <newline>, to the
 55050 standard output stream *stdout*. The terminating null byte shall not be written.

55051 CX The last data modification and last file status change timestamps of the file shall be marked for
 55052 update between the successful execution of *puts()* and the next successful completion of a call to
 55053 *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

55054 **RETURN VALUE**

55055 Upon successful completion, *puts()* shall return a non-negative number. Otherwise, it shall
 55056 CX return EOF, shall set an error indicator for the stream, and *errno* shall be set to indicate the error.

55057 **ERRORS**55058 Refer to *fputc()*.55059 **EXAMPLES**55060 **Printing to Standard Output**

55061 The following example gets the current time, converts it to a string using *localtime()* and
 55062 *asctime()*, and prints it to standard output using *puts()*. It then prints the number of minutes to
 55063 an event for which it is waiting.

```
55064 #include <time.h>
55065 #include <stdio.h>
55066 ...
55067 time_t now;
55068 int minutes_to_event;
55069 ...
55070 time(&now);
55071 printf("The time is ");
55072 puts(asctime(localtime(&now)));
55073 printf("There are %d minutes to the event.\n",
55074        minutes_to_event);
55075 ...
```

55076 **APPLICATION USAGE**55077 The *puts()* function appends a <newline>, while *fputs()* does not.55078 **RATIONALE**

55079 None.

puts()*System Interfaces*55080 **FUTURE DIRECTIONS**

55081 None.

55082 **SEE ALSO**55083 *fopen()*, *fputs()*, *putc()*55084 XBD **<stdio.h>**55085 **CHANGE HISTORY**

55086 First released in Issue 1. Derived from Issue 1 of the SVID.

55087 **Issue 6**

55088 Extensions beyond the ISO C standard are marked.

55089 **Issue 7**

55090 Changes are made related to support for finegrained timestamps.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***pututxline()**55091 **NAME**

55092 pututxline — put an entry into the user accounting database

55093 **SYNOPSIS**

55094 XSI #include <utmpx.h>

55095 struct utmpx *pututxline(const struct utmpx *utmpx);

55096 **DESCRIPTION**55097 Refer to *endutxent()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

putwc()*System Interfaces*55098 **NAME**55099 **putwc** — put a wide character on a stream55100 **SYNOPSIS**55101 `#include <stdio.h>`55102 `#include <wchar.h>`55103 `wint_t putwc(wchar_t wc, FILE *stream);`55104 **DESCRIPTION**

55105 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 55106 conflict between the requirements described here and the ISO C standard is unintentional. This
 55107 volume of POSIX.1-2008 defers to the ISO C standard.

55108 The *putwc()* function shall be equivalent to *fputwc()*, except that if it is implemented as a macro
 55109 it may evaluate *stream* more than once, so the argument should never be an expression with
 55110 side-effects.

55111 **RETURN VALUE**55112 Refer to *fputwc()*.55113 **ERRORS**55114 Refer to *fputwc()*.55115 **EXAMPLES**

55116 None.

55117 **APPLICATION USAGE**

55118 Since it may be implemented as a macro, *putwc()* may treat a *stream* argument with side-effects
 55119 incorrectly. In particular, *putwc(wc,*f++)* need not work correctly. Therefore, use of this function
 55120 is not recommended; *fputwc()* should be used instead.

55121 **RATIONALE**

55122 None.

55123 **FUTURE DIRECTIONS**

55124 None.

55125 **SEE ALSO**55126 *fputwc()*55127 XBD `<stdio.h>`, `<wchar.h>`55128 **CHANGE HISTORY**

55129 First released as a World-wide Portability Interface in Issue 4.

55130 **Issue 5**

55131 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*
 55132 is changed from **wint_t** to **wchar_t**.

55133 The Optional Header (OH) marking is removed from `<stdio.h>`.

55134 **NAME**

55135 putwchar — put a wide character on a stdout stream

55136 **SYNOPSIS**

55137 #include <wchar.h>

55138 wint_t putwchar(wchar_t wc);

55139 **DESCRIPTION**

55140 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 55141 conflict between the requirements described here and the ISO C standard is unintentional. This
 55142 volume of POSIX.1-2008 defers to the ISO C standard.

55143 The function call *putwchar(wc)* shall be equivalent to *putwc(wc, stdout)*.55144 **RETURN VALUE**55145 Refer to *fputwc()*.55146 **ERRORS**55147 Refer to *fputwc()*.55148 **EXAMPLES**

55149 None.

55150 **APPLICATION USAGE**

55151 None.

55152 **RATIONALE**

55153 None.

55154 **FUTURE DIRECTIONS**

55155 None.

55156 **SEE ALSO**55157 *fputwc()*, *putwc()*

55158 XBD <wchar.h>

55159 **CHANGE HISTORY**

55160 First released in Issue 4.

55161 **Issue 5**

55162 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*
 55163 is changed from **wint_t** to **wchar_t**.

pwrite()*System Interfaces*55164 **NAME**

55165 pwrite — write on a file

55166 **SYNOPSIS**

55167 #include <unistd.h>

55168 ssize_t pwrite(int *fildes*, const void **buf*, size_t *nbyte*,
55169 off_t *offset*);55170 **DESCRIPTION**55171 Refer to *write()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

qsort()**NAME**

qsort — sort a table of data

SYNOPSIS

```
#include <stdlib.h>

void qsort(void *base, size_t nel, size_t width,
           int (*compar)(const void *, const void *));
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *qsort()* function shall sort an array of *nel* objects, the initial element of which is pointed to by *base*. The size of each object, in bytes, is specified by the *width* argument. If the *nel* argument has the value zero, the comparison function pointed to by *compar* shall not be called and no rearrangement shall take place.

The application shall ensure that the comparison function pointed to by *compar* does not alter the contents of the array. The implementation may reorder elements of the array between calls to the comparison function, but shall not alter the contents of any individual element.

When the same objects (consisting of *width* bytes, irrespective of their current positions in the array) are passed more than once to the comparison function, the results shall be consistent with one another. That is, they shall define a total ordering on the array.

The contents of the array shall be sorted in ascending order according to a comparison function. The *compar* argument is a pointer to the comparison function, which is called with two arguments that point to the elements being compared. The application shall ensure that the function returns an integer less than, equal to, or greater than 0, if the first argument is considered respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is unspecified.

RETURN VALUE

The *qsort()* function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

RATIONALE

The requirement that each argument (hereafter referred to as *p*) to the comparison function is a pointer to elements of the array implies that for every call, for each argument separately, all of the following expressions are non-zero:

```
((char *)p - (char *)base) % width == 0
(char *)p >= (char *)base
(char *)p < (char *)base + nel * width
```

qsort()*System Interfaces*55214 **FUTURE DIRECTIONS**

55215 None.

55216 **SEE ALSO**55217 *alphasort()*55218 XBD *<stdlib.h>*55219 **CHANGE HISTORY**

55220 First released in Issue 1. Derived from Issue 1 of the SVID.

55221 **Issue 6**

55222 The normative text is updated to avoid use of the term “must” for application requirements.

55223 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/49 is applied, adding the last sentence to
55224 the first non-shaded paragraph in the DESCRIPTION, and the following two paragraphs. The
55225 RATIONALE is also updated. These changes are for alignment with the ISO C standard.

55226 NAME

55227 `raise` — send a signal to the executing process

55228 SYNOPSIS

55229 `#include <signal.h>`

55230 `int raise(int sig);`

55231 DESCRIPTION

55232 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 55233 conflict between the requirements described here and the ISO C standard is unintentional. This
 55234 volume of POSIX.1-2008 defers to the ISO C standard.

55235 CX The `raise()` function shall send the signal `sig` to the executing thread or process. If a signal
 55236 handler is called, the `raise()` function shall not return until after the signal handler does.

55237 CX The effect of the `raise()` function shall be equivalent to calling:

55238 `pthread_kill(pthread_self(), sig);`

55239 RETURN VALUE

55240 CX Upon successful completion, 0 shall be returned. Otherwise, a non-zero value shall be returned
 55241 and `errno` shall be set to indicate the error.

55242 ERRORS

55243 The `raise()` function shall fail if:

55244 CX [EINVAL] The value of the `sig` argument is an invalid signal number.

55245 EXAMPLES

55246 None.

55247 APPLICATION USAGE

55248 None.

55249 RATIONALE

55250 The term “thread” is an extension to the ISO C standard.

55251 FUTURE DIRECTIONS

55252 None.

55253 SEE ALSO

55254 [`kill\(\)`](#), [`sigaction\(\)`](#)

55255 XBD [`<signal.h>`](#), [`<sys/types.h>`](#)

55256 CHANGE HISTORY

55257 First released in Issue 4. Derived from the ANSI C standard.

55258 Issue 5

55259 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

55260 Issue 6

55261 Extensions beyond the ISO C standard are marked.

55262 The following new requirements on POSIX implementations derive from alignment with the
 55263 Single UNIX Specification:

- 55264 • In the RETURN VALUE section, the requirement to set `errno` on error is added.

raise()*System Interfaces*

55265

- The [EINVAL] error condition is added.

55266

Issue 7

55267

Functionality relating to the Threads option is moved to the Base.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

rand, rand_r, srand — pseudo-random number generator

SYNOPSIS

```
#include <stdlib.h>

int rand(void);
int rand_r(unsigned *seed);
void srand(unsigned seed);
```

DESCRIPTION

For *rand()* and *srand()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *rand()* function shall compute a sequence of pseudo-random integers in the range $[0, \{\text{RAND\_MAX}\}]$ with a period of at least 2^{32} .

The *rand()* function need not be thread-safe.

The *rand_r()* function shall compute a sequence of pseudo-random integers in the range $[0, \{\text{RAND\_MAX}\}]$. (The value of the $\{\text{RAND\_MAX}\}$ macro shall be at least 32767.)

If *rand_r()* is called with the same initial value for the object pointed to by *seed* and that object is not modified between successive returns and calls to *rand_r()*, the same sequence shall be generated.

The *srand()* function uses the argument as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to *rand()*. If *srand()* is then called with the same seed value, the sequence of pseudo-random numbers shall be repeated. If *rand()* is called before any calls to *srand()* are made, the same sequence shall be generated as when *srand()* is first called with a seed value of 1.

The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls *rand()* or *srand()*.

RETURN VALUE

The *rand()* function shall return the next pseudo-random number in the sequence.

The *rand_r()* function shall return a pseudo-random integer.

The *srand()* function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES**Generating a Pseudo-Random Number Sequence**

The following example demonstrates how to generate a sequence of pseudo-random numbers.

```
#include <stdio.h>
#include <stdlib.h>
...
    long count, i;
    char *keystr;
    int elementlen, len;
    char c;
...
```

rand()*System Interfaces*

```

55311      /* Initial random number generator. */
55312      srand(1);

55313      /* Create keys using only lowercase characters */
55314      len = 0;
55315      for (i=0; i<count; i++) {
55316          while (len < elementlen) {
55317              c = (char) (rand() % 128);
55318              if (islower(c))
55319                  keystr[len++] = c;
55320          }

55321          keystr[len] = '\\0';
55322          printf("%s Element%0*ld\\n", keystr, elementlen, i);
55323          len = 0;
55324      }

```

Generating the Same Sequence on Different Machines

The following code defines a pair of functions that could be incorporated into applications wishing to ensure that the same sequence of numbers is generated across different machines.

```

55328      static unsigned long next = 1;
55329      int myrand(void) /* RAND_MAX assumed to be 32767. */
55330      {
55331          next = next * 1103515245 + 12345;
55332          return((unsigned) (next/65536) % 32768);
55333      }

55334      void mysrand(unsigned seed)
55335      {
55336          next = seed;
55337      }

```

APPLICATION USAGE

The *drand48()* function provides a much more elaborate random number generator.

The limitations on the amount of state that can be carried between one function call and another mean the *rand_r()* function can never be implemented in a way which satisfies all of the requirements on a pseudo-random number generator. Therefore this function should be avoided whenever non-trivial requirements (including safety) have to be fulfilled.

RATIONALE

The ISO C standard *rand()* and *srand()* functions allow per-process pseudo-random streams shared by all threads. Those two functions need not change, but there has to be mutual-exclusion that prevents interference between two threads concurrently accessing the random number generator.

With regard to *rand()*, there are two different behaviors that may be wanted in a multi-threaded program:

1. A single per-process sequence of pseudo-random numbers that is shared by all threads that call *rand()*
2. A different sequence of pseudo-random numbers for each thread that calls *rand()*

This is provided by the modified thread-safe function based on whether the seed value is global

55355 to the entire process or local to each thread.

55356 This does not address the known deficiencies of the *rand()* function implementations, which
 55357 have been approached by maintaining more state. In effect, this specifies new thread-safe forms
 55358 of a deficient function.

55359 **FUTURE DIRECTIONS**

55360 The *rand_r()* function may be removed in a future version.

55361 **SEE ALSO**

55362 *drand48()*

55363 XBD `<stdlib.h>`

55364 **CHANGE HISTORY**

55365 First released in Issue 1. Derived from Issue 1 of the SVID.

55366 **Issue 5**

55367 The *rand_r()* function is included for alignment with the POSIX Threads Extension.

55368 A note indicating that the *rand()* function need not be reentrant is added to the DESCRIPTION.

55369 **Issue 6**

55370 Extensions beyond the ISO C standard are marked.

55371 The *rand_r()* function is marked as part of the Thread-Safe Functions option.

55372 **Issue 7**

55373 Austin Group Interpretation 1003.1-2001 #156 is applied.

55374 The *rand_r()* function is marked obsolescent.

random()*System Interfaces*55375 **NAME**

55376 random — generate pseudo-random number

55377 **SYNOPSIS**55378 XSI `#include <stdlib.h>`55379 `long random(void);`55380 **DESCRIPTION**55381 Refer to *initstate()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

read()**NAME**

`pread`, `read` — read from a file

SYNOPSIS

```
#include <unistd.h>

ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
ssize_t read(int fildes, void *buf, size_t nbyte);
```

DESCRIPTION

The `read()` function shall attempt to read *nbyte* bytes from the file associated with the open file descriptor, *fildes*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on the same pipe, FIFO, or terminal device is unspecified.

Before any action described below is taken, and if *nbyte* is zero, the `read()` function may detect and return errors as described below. In the absence of errors, or if error detection is not performed, the `read()` function shall return zero and have no other results.

On files that support seeking (for example, a regular file), the `read()` shall start at a position in the file given by the file offset associated with *fildes*. The file offset shall be incremented by the number of bytes actually read.

Files that do not support seeking—for example, terminals—always read from the current position. The value of a file offset associated with such a file is undefined.

No data transfer shall occur past the current end-of-file. If the starting position is at or after the end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent `read()` requests is implementation-defined.

If the value of *nbyte* is greater than {SSIZE_MAX}, the result is implementation-defined.

When attempting to read from an empty pipe or FIFO:

- If no process has the pipe open for writing, `read()` shall return 0 to indicate end-of-file.
- If some process has the pipe open for writing and `O_NONBLOCK` is set, `read()` shall return `-1` and set *errno* to `[EAGAIN]`.
- If some process has the pipe open for writing and `O_NONBLOCK` is clear, `read()` shall block the calling thread until some data is written or the pipe is closed by all processes that had the pipe open for writing.

When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and has no data currently available:

- If `O_NONBLOCK` is set, `read()` shall return `-1` and set *errno* to `[EAGAIN]`.
- If `O_NONBLOCK` is clear, `read()` shall block the calling thread until some data becomes available.
- The use of the `O_NONBLOCK` flag has no effect if there is some data available.

The `read()` function reads data previously written to a file. If any portion of a regular file prior to the end-of-file has not been written, `read()` shall return bytes with value 0. For example, `lseek()` allows the file offset to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads in the gap between the previous end of data and the newly written data shall return bytes with value 0 until data is written into the gap.

Upon successful completion, where *nbyte* is greater than 0, `read()` shall mark for update the last data access timestamp of the file, and shall return the number of bytes read. This number shall never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes

read()*System Interfaces*

55425		left in the file is less than <i>nbyte</i> , if the <i>read()</i> request was interrupted by a signal, or if the file is a
55426		pipe or FIFO or special file and has fewer than <i>nbyte</i> bytes immediately available for reading.
55427		For example, a <i>read()</i> from a file associated with a terminal may return one typed line of data.
55428		If a <i>read()</i> is interrupted by a signal before it reads any data, it shall return -1 with <i>errno</i> set to
55429		[EINTR].
55430		If a <i>read()</i> is interrupted by a signal after it has successfully read some data, it shall return the
55431		number of bytes read.
55432		For regular files, no data transfer shall occur past the offset maximum established in the open
55433		file description associated with <i>fildes</i> .
55434		If <i>fildes</i> refers to a socket, <i>read()</i> shall be equivalent to <i>recv()</i> with no flags set.
55435	SIO	If the O_DSYNC and O_RSYNC bits have been set, read I/O operations on the file descriptor
55436		shall complete as defined by synchronized I/O data integrity completion. If the O_SYNC and
55437		O_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as
55438		defined by synchronized I/O file integrity completion.
55439	SHM	If <i>fildes</i> refers to a shared memory object, the result of the <i>read()</i> function is unspecified.
55440	TYM	If <i>fildes</i> refers to a typed memory object, the result of the <i>read()</i> function is unspecified.
55441	OB XSR	A <i>read()</i> from a STREAMS file can read data in three different modes: <i>byte-stream</i> mode, <i>message-</i>
55442		<i>nondiscard</i> mode, and <i>message-discard</i> mode. The default shall be <i>byte-stream</i> mode. This can be
55443		changed using the <i>I_SRDOPT ioctl()</i> request, and can be tested with <i>I_GRDOPT ioctl()</i> . In <i>byte-</i>
55444		<i>stream</i> mode, <i>read()</i> shall retrieve data from the STREAM until as many bytes as were requested
55445		are transferred, or until there is no more data to be retrieved. <i>Byte-stream</i> mode ignores
55446		message boundaries.
55447		In STREAMS <i>message-nondiscard</i> mode, <i>read()</i> shall retrieve data until as many bytes as were
55448		requested are transferred, or until a message boundary is reached. If <i>read()</i> does not retrieve all
55449		the data in a message, the remaining data shall be left on the STREAM, and can be retrieved by
55450		the next <i>read()</i> call. <i>Message-discard</i> mode also retrieves data until as many bytes as were
55451		requested are transferred, or a message boundary is reached. However, unread data remaining
55452		in a message after the <i>read()</i> returns shall be discarded, and shall not be available for a
55453		subsequent <i>read()</i> , <i>getmsg()</i> , or <i>getpmsg()</i> call.
55454		How <i>read()</i> handles zero-byte STREAMS messages is determined by the current read mode
55455		setting. In <i>byte-stream</i> mode, <i>read()</i> shall accept data until it has read <i>nbyte</i> bytes, or until there
55456		is no more data to read, or until a zero-byte message block is encountered. The <i>read()</i> function
55457		shall then return the number of bytes read, and place the zero-byte message back on the
55458		STREAM to be retrieved by the next <i>read()</i> , <i>getmsg()</i> , or <i>getpmsg()</i> . In <i>message-nondiscard</i>
55459		mode or <i>message-discard</i> mode, a zero-byte message shall return 0 and the message shall be
55460		removed from the STREAM. When a zero-byte message is read as the first message on a
55461		STREAM, the message shall be removed from the STREAM and 0 shall be returned, regardless
55462		of the read mode.
55463		A <i>read()</i> from a STREAMS file shall return the data in the message at the front of the STREAM
55464		head read queue, regardless of the priority band of the message.
55465		By default, STREAMS are in <i>control-normal</i> mode, in which a <i>read()</i> from a STREAMS file can
55466		only process messages that contain a data part but do not contain a control part. The <i>read()</i> shall
55467		fail if a message containing a control part is encountered at the STREAM head. This default
55468		action can be changed by placing the STREAM in either <i>control-data</i> mode or <i>control-discard</i>
55469		mode with the <i>I_SRDOPT ioctl()</i> command. In <i>control-data</i> mode, <i>read()</i> shall convert any
55470		control part to data and pass it to the application before passing any data part originally present

55471 in the same message. In control-discard mode, *read()* shall discard message control parts but
 55472 return to the process any data part in the message.

55473 In addition, *read()* shall fail if the STREAM head had processed an asynchronous error before
 55474 the call. In this case, the value of *errno* shall not reflect the result of *read()*, but reflect the prior
 55475 error. If a hangup occurs on the STREAM being read, *read()* shall continue to operate normally
 55476 until the STREAM head read queue is empty. Thereafter, it shall return 0.

55477 The *pread()* function shall be equivalent to *read()*, except that it shall read from a given position
 55478 in the file without changing the file pointer. The first three arguments to *pread()* are the same as
 55479 *read()* with the addition of a fourth argument *offset* for the desired position inside the file. An
 55480 attempt to perform a *pread()* on a file that is incapable of seeking shall result in an error.

55481 RETURN VALUE

55482 Upon successful completion, these functions shall return a non-negative integer indicating the
 55483 number of bytes actually read. Otherwise, the functions shall return -1 and set *errno* to indicate
 55484 the error.

55485 ERRORS

55486 These functions shall fail if:

55487 [EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the thread would be
 55488 delayed.

55489 [EBADF] The *fildev* argument is not a valid file descriptor open for reading.

55490 OB XSR [EBADMSG] The file is a STREAM file that is set to control-normal mode and the message
 55491 waiting to be read includes a control part.

55492 [EINTR] The read operation was terminated due to the receipt of a signal, and no data
 55493 was transferred.

55494 OB XSR [EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or
 55495 indirectly) downstream from a multiplexer.

55496 [EIO] The process is a member of a background process group attempting to read
 55497 from its controlling terminal, the process is ignoring or blocking the SIGTTIN
 55498 signal, or the process group is orphaned. This error may also be generated for
 55499 implementation-defined reasons.

55500 XSI [EISDIR] The *fildev* argument refers to a directory and the implementation does not
 55501 allow the directory to be read using *read()* or *pread()*. The *readdir()* function
 55502 should be used instead.

55503 [EOVERFLOW] The file is a regular file, *nbyte* is greater than 0, the starting position is before
 55504 the end-of-file, and the starting position is greater than or equal to the offset
 55505 maximum established in the open file description associated with *fildev*.

55506 The *read()* function shall fail if:

55507 [EAGAIN] or [EWOULDBLOCK]

55508 The file descriptor is for a socket, is marked O_NONBLOCK, and no data is
 55509 waiting to be received.

55510 [ECONNRESET] A read was attempted on a socket and the connection was forcibly closed by
 55511 its peer.

55512 [ENOTCONN] A read was attempted on a socket that is not connected.

read()*System Interfaces*

- 55513 [ETIMEDOUT] A read was attempted on a socket and a transmission timeout occurred.
- 55514 These functions may fail if:
- 55515 [EIO] A physical I/O error has occurred.
- 55516 [ENOBUFFS] Insufficient resources were available in the system to perform the operation.
- 55517 [ENOMEM] Insufficient memory was available to fulfill the request.
- 55518 [ENXIO] A request was made of a nonexistent device, or the request was outside the
- 55519 capabilities of the device.
- 55520 The *pread()* function shall fail, and the file pointer shall remain unchanged, if:
- 55521 [EINVAL] The *offset* argument is invalid. The value is negative.
- 55522 [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the
- 55523 offset maximum associated with the file.
- 55524 [ENXIO] A request was outside the capabilities of the device.
- 55525 [ESPIPE] *fildev* is associated with a pipe or FIFO.

55526 **EXAMPLES**55527 **Reading Data into a Buffer**

55528 The following example reads data from the file associated with the file descriptor *fd* into the

55529 buffer pointed to by *buf*.

```

55530 #include <sys/types.h>
55531 #include <unistd.h>
55532 ...
55533 char buf[20];
55534 size_t nbytes;
55535 ssize_t bytes_read;
55536 int fd;
55537 ...
55538 nbytes = sizeof(buf);
55539 bytes_read = read(fd, buf, nbytes);
55540 ...

```

55541 **APPLICATION USAGE**

55542 None.

55543 **RATIONALE**

55544 This volume of POSIX.1-2008 does not specify the value of the file offset after an error is

55545 returned; there are too many cases. For programming errors, such as [EBADF], the concept is

55546 meaningless since no file is involved. For errors that are detected immediately, such as

55547 [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however,

55548 an updated value would be very useful and is the behavior of many implementations.

55549 Note that a *read()* of zero bytes does not modify the last data access timestamp. A *read()* that

55550 requests more than zero bytes, but returns zero, is required to modify the last data access

55551 timestamp.

55552 Implementations are allowed, but not required, to perform error checking for *read()* requests of

55553 zero bytes.

Input and Output

The use of I/O with large byte counts has always presented problems. Ideas such as *lread()* and *lwrite()* (using and returning **longs**) were considered at one time. The current solution is to use abstract types on the ISO C standard function to *read()* and *write()*. The abstract types can be declared so that existing functions work, but can also be declared so that larger types can be represented in future implementations. It is presumed that whatever constraints limit the maximum range of **size_t** also limit portable I/O requests to the same range. This volume of POSIX.1-2008 also limits the range further by requiring that the byte count be limited so that a signed return value remains meaningful. Since the return type is also a (signed) abstract type, the byte count can be defined by the implementation to be larger than an **int** can hold.

The standard developers considered adding atomicity requirements to a pipe or FIFO, but recognized that due to the nature of pipes and FIFOs there could be no guarantee of atomicity of reads of {PIPE_BUF} or any other size that would be an aid to applications portability.

This volume of POSIX.1-2008 requires that no action be taken for *read()* or *write()* when *nbyte* is zero. This is not intended to take precedence over detection of errors (such as invalid buffer pointers or file descriptors). This is consistent with the rest of this volume of POSIX.1-2008, but the phrasing here could be misread to require detection of the zero case before any other errors. A value of zero is to be considered a correct value, for which the semantics are a no-op.

I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the bytes from a single operation that started out together end up together, without interleaving from other I/O operations. It is a known attribute of terminals that this is not honored, and terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified. The behavior for other device types is also left unspecified, but the wording is intended to imply that future standards might choose to specify atomicity (or not).

There were recommendations to add format parameters to *read()* and *write()* in order to handle networked transfers among heterogeneous file system and base hardware types. Such a facility may be required for support by the OSI presentation of layer services. However, it was determined that this should correspond with similar C-language facilities, and that is beyond the scope of this volume of POSIX.1-2008. The concept was suggested to the developers of the ISO C standard for their consideration as a possible area for future work.

In 4.3 BSD, a *read()* or *write()* that is interrupted by a signal before transferring any data does not by default return an [EINTR] error, but is restarted. In 4.2 BSD, 4.3 BSD, and the Eighth Edition, there is an additional function, *select()*, whose purpose is to pause until specified activity (data to read, space to write, and so on) is detected on specified file descriptors. It is common in applications written for those systems for *select()* to be used before *read()* in situations (such as keyboard input) where interruption of I/O due to a signal is desired.

The issue of which files or file types are interruptible is considered an implementation design issue. This is often affected primarily by hardware and reliability issues.

There are no references to actions taken following an “unrecoverable error”. It is considered beyond the scope of this volume of POSIX.1-2008 to describe what happens in the case of hardware errors.

Earlier versions of this standard allowed two very different behaviors with regard to the handling of interrupts. In order to minimize the resulting confusion, it was decided that POSIX.1-2008 should support only one of these behaviors. Historical practice on AT&T-derived systems was to have *read()* and *write()* return -1 and set *errno* to [EINTR] when interrupted after some, but not all, of the data requested had been transferred. However, the US Department of Commerce FIPS 151-1 and FIPS 151-2 require the historical BSD behavior, in which *read()* and

read()*System Interfaces*

write() return the number of bytes actually transferred before the interrupt. If `-1` is returned when any data is transferred, it is difficult to recover from the error on a seekable device and impossible on a non-seekable device. Most new implementations support this behavior. The behavior required by POSIX.1-2008 is to return the number of bytes transferred.

POSIX.1-2008 does not specify when an implementation that buffers *read()*s actually moves the data into the user-supplied buffer, so an implementation may choose to do this at the latest possible moment. Therefore, an interrupt arriving earlier may not cause *read()* to return a partial byte count, but rather to return `-1` and set *errno* to `[EINTR]`.

Consideration was also given to combining the two previous options, and setting *errno* to `[EINTR]` while returning a short count. However, not only is there no existing practice that implements this, it is also contradictory to the idea that when *errno* is set, the function responsible shall return `-1`.

FUTURE DIRECTIONS

None.

SEE ALSO

fcntl(), *ioctl()*, *lseek()*, *open()*, *pipe()*, *readv()*

XBD Chapter 11 (on page 199), `<stropts.h>`, `<sys/uio.h>`, `<unistd.h>`

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

The *pread()* function is added.

Issue 6

The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION now states that if *read()* is interrupted by a signal after it has successfully read some data, it returns the number of bytes read. In Issue 3, it was optional whether *read()* returned the number of bytes read, or whether it returned `-1` with *errno* set to `[EINTR]`. This is a FIPS requirement.
- In the DESCRIPTION, text is added to indicate that for regular files, no data transfer occurs past the offset maximum established in the open file description associated with *fildev*. This change is to support large files.
- The `[EOVERFLOW]` mandatory error condition is added.
- The `[ENXIO]` optional error condition is added.

Text referring to sockets is added to the DESCRIPTION.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effect of reading zero bytes is clarified.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that

- 55643 *read()* results are unspecified for typed memory objects.
- 55644 New RATIONALE is added to explain the atomicity requirements for input and output
55645 operations.
- 55646 The following error conditions are added for operations on sockets: [EAGAIN],
55647 [ECONNRESET], [ENOTCONN], and [ETIMEDOUT].
- 55648 The [EIO] error is made optional.
- 55649 The following error conditions are added for operations on sockets: [ENOBUFFS] and
55650 [ENOMEM].
- 55651 The *readv()* function is split out into a separate reference page.
- 55652 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/108 is applied, updating the [EAGAIN]
55653 error in the ERRORS section from “the process would be delayed” to “the thread would be
55654 delayed”.
- 55655 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/109 is applied, making an editorial
55656 correction in the RATIONALE section.
- 55657 **Issue 7**
- 55658 The *pread()* function is moved from the XSI option to the Base.
- 55659 Functionality relating to the XSI STREAMS option is marked obsolescent.
- 55660 Changes are made related to support for finegrained timestamps.

readdir()*System Interfaces*55661 **NAME**

55662 readdir, readdir_r — read a directory

55663 **SYNOPSIS**

```

55664       #include <dirent.h>

55665       struct dirent *readdir(DIR *dirp);
55666       int readdir_r(DIR *restrict dirp, struct dirent *restrict entry,
55667                    struct dirent **restrict result);

```

55668 **DESCRIPTION**

55669 The type **DIR**, which is defined in the **<dirent.h>** header, represents a *directory stream*, which is
 55670 an ordered sequence of all the directory entries in a particular directory. Directory entries
 55671 represent files; files may be removed from a directory or added to a directory asynchronously to
 55672 the operation of *readdir()*.

55673 The *readdir()* function shall return a pointer to a structure representing the directory entry at the
 55674 current position in the directory stream specified by the argument *dirp*, and position the
 55675 directory stream at the next entry. It shall return a null pointer upon reaching the end of the
 55676 directory stream. The structure **dirent** defined in the **<dirent.h>** header describes a directory
 55677 entry. The value of the structure's *d_ino* member shall be set to the file serial number of the file
 55678 named by the *d_name* member. If the *d_name* member names a symbolic link, the value of the
 55679 *d_ino* member shall be set to the file serial number of the symbolic link itself.

55680 The *readdir()* function shall not return directory entries containing empty names. If entries for
 55681 dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-
 55682 dot; otherwise, they shall not be returned.

55683 The pointer returned by *readdir()* points to data which may be overwritten by another call to
 55684 *readdir()* on the same directory stream. This data is not overwritten by another call to *readdir()*
 55685 on a different directory stream.

55686 If a file is removed from or added to the directory after the most recent call to *opendir()* or
 55687 *rewinddir()*, whether a subsequent call to *readdir()* returns an entry for that file is unspecified.

55688 The *readdir()* function may buffer several directory entries per actual read operation; *readdir()*
 55689 shall mark for update the last data access timestamp of the directory each time the directory is
 55690 actually read.

55691 After a call to *fork()*, either the parent or child (but not both) may continue processing the
 55692 directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and child processes
 55693 use these functions, the result is undefined.

55694 The *readdir()* function need not be thread-safe.

55695 Applications wishing to check for error situations should set *errno* to 0 before calling *readdir()*. If
 55696 *errno* is set to non-zero on return, an error occurred.

55697 The *readdir_r()* function shall initialize the **dirent** structure referenced by *entry* to represent the
 55698 directory entry at the current position in the directory stream referred to by *dirp*, store a pointer
 55699 to this structure at the location referenced by *result*, and position the directory stream at the next
 55700 entry.

55701 The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d_name*
 55702 members containing at least [NAME_MAX]+1 elements.

55703 Upon successful return, the pointer returned at **result* shall have the same value as the argument
 55704 *entry*. Upon reaching the end of the directory stream, this pointer shall have the value NULL.

55705 The *readdir_r()* function shall not return directory entries containing empty names.

If a file is removed from or added to the directory after the most recent call to *opendir()* or *rewinddir()*, whether a subsequent call to *readdir_r()* returns an entry for that file is unspecified.

The *readdir_r()* function may buffer several directory entries per actual read operation; *readdir_r()* shall mark for update the last data access timestamp of the directory each time the directory is actually read.

RETURN VALUE

Upon successful completion, *readdir()* shall return a pointer to an object of type **struct dirent**. When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate the error. When the end of the directory is encountered, a null pointer shall be returned and *errno* is not changed.

If successful, the *readdir_r()* function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall fail if:

[EOVERFLOW] One of the values in the structure to be returned cannot be represented correctly.

These functions may fail if:

[EBADF] The *dirp* argument does not refer to an open directory stream.

[ENOENT] The current position of the directory stream is invalid.

EXAMPLES

The following sample program searches the current directory for each of the arguments supplied on the command line.

```
#include <dirent.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>

static void lookup(const char *arg)
{
    DIR *dirp;
    struct dirent *dp;

    if ((dirp = opendir(".")) == NULL) {
        perror("couldn't open '.'");
        return;
    }

    do {
        errno = 0;
        if ((dp = readdir(dirp)) != NULL) {
            if (strcmp(dp->d_name, arg) != 0)
                continue;

            (void) printf("found %s\n", arg);
            (void) closedir(dirp);
            return;
        }
    } while (dp != NULL);
}
```

readdir()*System Interfaces*

```

55750         if (errno != 0)
55751             perror("error reading directory");
55752         else
55753             (void) printf("failed to find %s\n", arg);
55754         (void) closedir(dirp);
55755         return;
55756     }

55757     int main(int argc, char *argv[])
55758     {
55759         int i;
55760         for (i = 1; i < argc; i++)
55761             lookup(argv[i]);
55762         return (0);
55763     }

```

APPLICATION USAGE

The *readdir()* function should be used in conjunction with *opendir()*, *closedir()*, and *rewinddir()* to examine the contents of the directory.

The *readdir_r()* function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

The returned value of *readdir()* merely *represents* a directory entry. No equivalence should be inferred.

Historical implementations of *readdir()* obtain multiple directory entries on a single read operation, which permits subsequent *readdir()* operations to operate from the buffered information. Any wording that required each successful *readdir()* operation to mark the directory last data access timestamp for update would disallow such historical performance-oriented implementations.

When returning a directory entry for the root of a mounted file system, some historical implementations of *readdir()* returned the file serial number of the underlying mount point, rather than of the root of the mounted file system. This behavior is considered to be a bug, since the underlying file serial number has no significance to applications.

Since *readdir()* returns NULL when it detects an error and when the end of the directory is encountered, an application that needs to tell the difference must set *errno* to zero before the call and check it if NULL is returned. Since the function must not change *errno* in the second case and must set it to a non-zero value in the first case, a zero *errno* after a call returning NULL indicates end-of-directory; otherwise, an error.

Routines to deal with this problem more directly were proposed:

```

55787     int derror (dirp)
55788     DIR *dirp;

55789     void clearderr (dirp)
55790     DIR *dirp;

```

The first would indicate whether an error had occurred, and the second would clear the error indication. The simpler method involving *errno* was adopted instead by requiring that *readdir()* not change *errno* when end-of-directory is encountered.

An error or signal indicating that a directory has changed while open was considered but rejected.

55796 The thread-safe version of the directory reading function returns values in a user-supplied buffer
 55797 instead of possibly using a static data area that may be overwritten by each call. Either the
 55798 {NAME_MAX} compile-time constant or the corresponding *pathconf()* option can be used to
 55799 determine the maximum sizes of returned pathnames.

55800 FUTURE DIRECTIONS

55801 None.

55802 SEE ALSO

55803 *closedir()*, *dirfd()*, *exec*, *fdopendir()*, *fstatat()*, *rewinddir()*, *symlink()*

55804 XBD **<dirent.h>**, **<sys/types.h>**

55805 CHANGE HISTORY

55806 First released in Issue 2.

55807 Issue 5

55808 Large File Summit extensions are added.

55809 The *readdir_r()* function is included for alignment with the POSIX Threads Extension.

55810 A note indicating that the *readdir()* function need not be reentrant is added to the
 55811 DESCRIPTION.

55812 Issue 6

55813 The *readdir_r()* function is marked as part of the Thread-Safe Functions option.

55814 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r()*.

55815 The Open Group Corrigendum U026/8 is applied, clarifying the wording of the successful
 55816 return for the *readdir_r()* function.

55817 The following new requirements on POSIX implementations derive from alignment with the
 55818 Single UNIX Specification:

- 55819 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
 55820 required for conforming implementations of previous POSIX specifications, it was not
 55821 required for UNIX applications.
- 55822 • A statement is added to the DESCRIPTION indicating the disposition of certain fields in
 55823 **struct dirent** when an entry refers to a symbolic link.
- 55824 • The [EOVERFLOW] mandatory error condition is added. This change is to support large
 55825 files.
- 55826 • The [ENOENT] optional error condition is added.

55827 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
 55828 its avoidance of possibly using a static data area.

55829 The **restrict** keyword is added to the *readdir_r()* prototype for alignment with the
 55830 ISO/IEC 9899:1999 standard.

55831 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/50 is applied, replacing the EXAMPLES
 55832 section with a new example.

55833 Issue 7

55834 Austin Group Interpretation 1003.1-2001 #059 is applied, updating the ERRORS section.

55835 Austin Group Interpretation 1003.1-2001 #156 is applied.

55836 The *readdir_r()* function is moved from the Thread-Safe Functions option to the Base.

readdir()*System Interfaces*

55837	Changes are made related to support for finegrained timestamps.
55838	The value of the <i>d_ino</i> member is no longer unspecified for symbolic links.
55839	SD5-XSH-ERN-193 is applied.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

readlink, readlinkat — read the contents of a symbolic link relative to a directory file descriptor

SYNOPSIS

```
#include <unistd.h>

ssize_t readlink(const char *restrict path, char *restrict buf,
                 size_t bufsize);
ssize_t readlinkat(int fd, const char *restrict path,
                  char *restrict buf, size_t bufsize);
```

DESCRIPTION

The *readlink()* function shall place the contents of the symbolic link referred to by *path* in the buffer *buf* which has size *bufsize*. If the number of bytes in the symbolic link is less than *bufsize*, the contents of the remainder of *buf* are unspecified. If the *buf* argument is not large enough to contain the link content, the first *bufsize* bytes shall be placed in *buf*.

If the value of *bufsize* is greater than {SSIZE_MAX}, the result is implementation-defined.

Upon successful completion, *readlink()* shall mark for update the last data access timestamp of the symbolic link.

The *readlinkat()* function shall be equivalent to the *readlink()* function except in the case where *path* specifies a relative path. In this case the symbolic link whose content is read is relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function shall not perform the check.

If *readlinkat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working directory is used and the behavior shall be identical to a call to *readlink()*.

RETURN VALUE

Upon successful completion, *readlink()* shall return the count of bytes placed in the buffer. Otherwise, it shall return a value of -1, leave the buffer unchanged, and set *errno* to indicate the error.

Upon successful completion, the *readlinkat()* function shall return 0. Otherwise, it shall return -1 and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

[EACCESS] Search permission is denied for a component of the path prefix of *path*.

[EINVAL] The *path* argument names a file that is not a symbolic link.

[EIO] An I/O error occurred while reading from the file system.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix is not a directory, or the *path* argument contains at least one non-*<slash>* character and ends with one or more trailing *<slash>* characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

readlink()*System Interfaces*

- 55884 The *readlinkat()* function shall fail if:
- 55885 [EACCES] *fd* was not opened with O_SEARCH and the permissions of the directory
55886 underlying *fd* do not permit directory searches.
- 55887 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
55888 neither AT_FDCWD nor a valid file descriptor open for reading or searching.
- 55889 These functions may fail if:
- 55890 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
55891 resolution of the *path* argument.
- 55892 [ENAMETOOLONG]
55893 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
55894 symbolic link produced an intermediate result with a length that exceeds
55895 {PATH_MAX}.
- 55896 The *readlinkat()* function may fail if:
- 55897 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
55898 file descriptor associated with a directory.

EXAMPLES**Reading the Name of a Symbolic Link**

55900 The following example shows how to read the name of a symbolic link named **/modules/pass1**.

```
55901 #include <unistd.h>
55902
55903 char buf[1024];
55904 ssize_t len;
55905 ...
55906 if ((len = readlink("/modules/pass1", buf, sizeof(buf)-1)) != -1)
55907     buf[len] = '\0';
```

APPLICATION USAGE

55908 Conforming applications should not assume that the returned contents of the symbolic link are
55909 null-terminated.

RATIONALE

55912 Since POSIX.1-2008 does not require any association of file times with symbolic links, there is no
55913 requirement that file times be updated by *readlink()*. The type associated with *bufsiz* is a **size_t**
55914 in order to be consistent with both the ISO C standard and the definition of *read()*. The behavior
55915 specified for *readlink()* when *bufsiz* is zero represents historical practice. For this case, the
55916 standard developers considered a change whereby *readlink()* would return the number of non-
55917 null bytes contained in the symbolic link with the buffer *buf* remaining unchanged; however,
55918 since the **stat** structure member *st_size* value can be used to determine the size of buffer
55919 necessary to contain the contents of the symbolic link as returned by *readlink()*, this proposal
55920 was rejected, and the historical practice retained.

55921 The purpose of the *readlinkat()* function is to read the content of symbolic links in directories
55922 other than the current working directory without exposure to race conditions. Any part of the
55923 path of a file could be changed in parallel to a call to *readlink()*, resulting in unspecified behavior.
55924 By opening a file descriptor for the target directory and using the *readlinkat()* function it can be
55925 guaranteed that the symbolic link read is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO*fstatat()*, *symlink()*XBD `<unistd.h>`**CHANGE HISTORY**

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6The return type is changed to **ssize_t**, to align with the IEEE P1003.1a draft standard.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- This function is made mandatory.
- In this function it is possible for the return value to exceed the range of the type **ssize_t** (since **size_t** has a larger range of positive values than **ssize_t**). A sentence restricting the size of the **size_t** object is added to the description to resolve this conflict.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The FUTURE DIRECTIONS section is changed to None.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

The **restrict** keyword is added to the *readlink()* prototype for alignment with the ISO/IEC 9899: 1999 standard.**Issue 7**

Austin Group Interpretation 1003.1-2001 #143 is applied.

SD5-XSH-ERN-189 is applied, updating the ERRORS section.

The *readlinkat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The [EACCES] error is removed from the “may fail” error conditions.

Changes are made to allow a directory to be opened for searching.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

readv()*System Interfaces*55958 **NAME**

55959 readv — read a vector

55960 **SYNOPSIS**

```
55961 XSI      #include <sys/uio.h>
55962         ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);
```

55963 **DESCRIPTION**

55964 The *readv()* function shall be equivalent to *read()*, except as described below. The *readv()*
 55965 function shall place the input data into the *iovcnt* buffers specified by the members of the *iov*
 55966 array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*−1]. The *iovcnt* argument is valid if greater than 0 and less than
 55967 or equal to {IOV_MAX}.

55968 Each *iovec* entry specifies the base address and length of an area in memory where data should
 55969 be placed. The *readv()* function shall always fill an area completely before proceeding to the
 55970 next.

55971 Upon successful completion, *readv()* shall mark for update the last data access timestamp of the
 55972 file.

55973 **RETURN VALUE**55974 Refer to *read()*.55975 **ERRORS**55976 Refer to *read()*.55977 In addition, the *readv()* function shall fail if:

55978 [EINVAL] The sum of the *iov_len* values in the *iov* array overflowed an **ssize_t**.

55979 The *readv()* function may fail if:

55980 [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.

55981 **EXAMPLES**55982 **Reading Data into an Array**

55983 The following example reads data from the file associated with the file descriptor *fd* into the
 55984 buffers specified by members of the *iov* array.

```
55985 #include <sys/types.h>
55986 #include <sys/uio.h>
55987 #include <unistd.h>
55988 ...
55989 ssize_t bytes_read;
55990 int fd;
55991 char buf0[20];
55992 char buf1[30];
55993 char buf2[40];
55994 int iovcnt;
55995 struct iovec iov[3];

55996 iov[0].iov_base = buf0;
55997 iov[0].iov_len = sizeof(buf0);
55998 iov[1].iov_base = buf1;
55999 iov[1].iov_len = sizeof(buf1);
56000 iov[2].iov_base = buf2;
```

System Interfaces

readv()

```

56001         iov[2].iov_len = sizeof(buf2);
56002         ...
56003         iovcnt = sizeof(iov) / sizeof(struct iovec);
56004         bytes_read = readv(fd, iov, iovcnt);
56005         ...

```

56006 **APPLICATION USAGE**

56007 None.

56008 **RATIONALE**56009 Refer to *read()*.56010 **FUTURE DIRECTIONS**

56011 None.

56012 **SEE ALSO**56013 *read()*, *writev()*56014 XBD [<sys/uio.h>](#)56015 **CHANGE HISTORY**

56016 First released in Issue 4, Version 2.

56017 **Issue 6**56018 Split out from the *read()* reference page.56019 **Issue 7**

56020 Changes are made related to support for finegrained timestamps.

realloc()*System Interfaces*56021 **NAME**56022 **realloc** — memory reallocator56023 **SYNOPSIS**56024 `#include <stdlib.h>`56025 `void *realloc(void *ptr, size_t size);`56026 **DESCRIPTION**

56027 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 56028 conflict between the requirements described here and the ISO C standard is unintentional. This
 56029 volume of POSIX.1-2008 defers to the ISO C standard.

56030 The *realloc()* function shall change the size of the memory object pointed to by *ptr* to the size
 56031 specified by *size*. The contents of the object shall remain unchanged up to the lesser of the new
 56032 and old sizes. If the new size of the memory object would require movement of the object, the
 56033 space for the previous instantiation of the object is freed. If the new size is larger, the contents of
 56034 the newly allocated portion of the object are unspecified. If *size* is 0 and *ptr* is not a null pointer,
 56035 the object pointed to is freed. If the space cannot be allocated, the object shall remain unchanged.

56036 If *ptr* is a null pointer, *realloc()* shall be equivalent to *malloc()* for the specified size.

56037 If *ptr* does not match a pointer returned earlier by *calloc()*, *malloc()*, or *realloc()* or if the space has
 56038 previously been deallocated by a call to *free()* or *realloc()*, the behavior is undefined.

56039 The order and contiguity of storage allocated by successive calls to *realloc()* is unspecified. The
 56040 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
 56041 a pointer to any type of object and then used to access such an object in the space allocated (until
 56042 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object
 56043 disjoint from any other object. The pointer returned shall point to the start (lowest byte address)
 56044 of the allocated space. If the space cannot be allocated, a null pointer shall be returned.

56045 **RETURN VALUE**

56046 Upon successful completion with a size not equal to 0, *realloc()* shall return a pointer to the
 56047 (possibly moved) allocated space. If *size* is 0, either a null pointer or a unique pointer that can be
 56048 successfully passed to *free()* shall be returned. If there is not enough available memory, *realloc()*
 56049 CX shall return a null pointer and set *errno* to [ENOMEM].

56050 **ERRORS**

56051 The *realloc()* function shall fail if:

56052 CX [ENOMEM] Insufficient memory is available.

56053 **EXAMPLES**

56054 None.

56055 **APPLICATION USAGE**

56056 None.

56057 **RATIONALE**

56058 None.

56059 **FUTURE DIRECTIONS**

56060 None.

56061 **SEE ALSO**

56062 *calloc()*, *free()*, *malloc()*

56063 XBD <stdlib.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, if there is not enough available memory, the setting of *errno* to [ENOMEM] is added.
- The [ENOMEM] error condition is added.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

realpath()*System Interfaces*56073 **NAME**

56074 realpath — resolve a pathname

56075 **SYNOPSIS**

```
56076 XSI      #include <stdlib.h>
56077
56077 char *realpath(const char *restrict file_name,
56078               char *restrict resolved_name);
```

56079 **DESCRIPTION**

56080 The *realpath()* function shall derive, from the pathname pointed to by *file_name*, an absolute
 56081 pathname that resolves to the same directory entry, whose resolution does not involve '.',
 56082 '..', or symbolic links. If *resolved_name* is a null pointer, the generated pathname shall be
 56083 stored as a null-terminated string in a buffer allocated as if by a call to *malloc()*. Otherwise, if
 56084 {PATH_MAX} is defined as a constant in the <limits.h> header, then the generated pathname
 56085 shall be stored as a null-terminated string, up to a maximum of {PATH_MAX} bytes, in the
 56086 buffer pointed to by *resolved_name*.

56087 If *resolved_name* is not a null pointer and {PATH_MAX} is not defined as a constant in the
 56088 <limits.h> header, the behavior is undefined.

56089 **RETURN VALUE**

56090 Upon successful completion, *realpath()* shall return a pointer to the buffer containing the
 56091 resolved name. Otherwise, *realpath()* shall return a null pointer and set *errno* to indicate the
 56092 error.

56093 If the *resolved_name* argument is a null pointer, the pointer returned by *realpath()* can be passed
 56094 to *free()*.

56095 If the *resolved_name* argument is not a null pointer and the *realpath()* function fails, the contents
 56096 of the buffer pointed to by *resolved_name* are undefined.

56097 **ERRORS**

56098 The *realpath()* function shall fail if:

56099	[EACCES]	Read or search permission was denied for a component of <i>file_name</i> .
56100	[EINVAL]	The <i>file_name</i> argument is a null pointer.
56101	[EIO]	An error occurred while reading from the file system.
56102	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>file_name</i> argument.
56103		
56104	[ENAMETOOLONG]	
56105		The length of a component of a pathname is longer than {NAME_MAX}.
56106	[ENOENT]	A component of <i>file_name</i> does not name an existing file or <i>file_name</i> points to an empty string.
56107		
56108	[ENOTDIR]	A component of the path prefix is not a directory, or the <i>file_name</i> argument contains at least one non- <i><slash></i> character and ends with one or more trailing <i><slash></i> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
56109		
56110		
56111		

56112 The *realpath()* function may fail if:

56113	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>file_name</i> argument.
56114		

[ENAMETOOLONG]

The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[ENOMEM]

Insufficient storage space is available.

EXAMPLES**Generating an Absolute Pathname**

The following example generates an absolute pathname for the file identified by the *symlinkpath* argument. The generated pathname is stored in the buffer pointed to by *actualpath*.

```
#include <stdlib.h>
...
char *symlinkpath = "/tmp/symlink/file";
char *actualpath;

actualpath = realpath(symlinkpath, NULL);
if (actualpath != NULL)
{
    ... use actualpath ...

    free(actualpath);
}
else
{
    ... handle error ...
}
```

APPLICATION USAGE

For functions that allocate memory as if by *malloc()*, the application should release such memory when it is no longer required by a call to *free()*. For *realpath()*, this is the return value.

RATIONALE

Since *realpath()* has no *length* argument, if {PATH_MAX} is not defined as a constant in **<limits.h>**, applications have no way of determining how large a buffer they need to allocate for it to be safe to pass to *realpath()*. A {PATH_MAX} value obtained from a prior *pathconf()* call is out-of-date by the time *realpath()* is called. Hence the only reliable way to use *realpath()* when {PATH_MAX} is not defined in **<limits.h>** is to pass a null pointer for *resolved_name* so that *realpath()* will allocate a buffer of the necessary size.

FUTURE DIRECTIONS

None.

SEE ALSO

fpathconf(), *free()*, *getcwd()*, *sysconf()*

XBD **<limits.h>**, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

realpath()*System Interfaces*56157 **Issue 6**

56158 The **restrict** keyword is added to the *realpath()* prototype for alignment with the
56159 ISO/IEC 9899:1999 standard.

56160 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
56161 [ELOOP] error condition is added.

56162 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/51 is applied, adding new text to the
56163 DESCRIPTION for the case when *resolved_name* is a null pointer, changing the [EINVAL] error
56164 text, adding text to the RATIONALE, and adding text to FUTURE DIRECTIONS.

56165 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/110 is applied, updating the ERRORS
56166 section to refer to the *file_name* argument, rather than a nonexistent *path* argument.

56167 **Issue 7**

56168 Austin Group Interpretation 1003.1-2001 #143 is applied.

56169 This function is updated for passing a null pointer to *realpath()* for the *resolved_name* argument.

56170 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by
56171 *malloc()*.

System Interfaces

recv()**NAME**

`recv` — receive a message from a connected socket

SYNOPSIS

```
#include <sys/socket.h>
```

```
ssize_t recv(int socket, void *buffer, size_t length, int flags);
```

DESCRIPTION

The `recv()` function shall receive a message from a connection-mode or connectionless-mode socket. It is normally used with connected sockets because it does not permit the application to retrieve the source address of received data.

The `recv()` function takes the following arguments:

socket Specifies the socket file descriptor.

buffer Points to a buffer where the message should be stored.

length Specifies the length in bytes of the buffer pointed to by the *buffer* argument.

flags Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:

`MSG_PEEK` Peeks at an incoming message. The data is treated as unread and the next `recv()` or similar function shall still return this data.

`MSG_OOB` Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.

`MSG_WAITALL` On `SOCK_STREAM` sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if `MSG_PEEK` was specified, or if an error is pending for the socket.

The `recv()` function shall return the length of the message written to the buffer pointed to by the *buffer* argument. For message-based sockets, such as `SOCK_DGRAM` and `SOCK_SEQPACKET`, the entire message shall be read in a single operation. If a message is too long to fit in the supplied buffer, and `MSG_PEEK` is not set in the *flags* argument, the excess bytes shall be discarded. For stream-based sockets, such as `SOCK_STREAM`, message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes available, and no data shall be discarded.

If the `MSG_WAITALL` flag is not set, data shall be returned only up to the end of the first message.

If no messages are available at the socket and `O_NONBLOCK` is not set on the socket's file descriptor, `recv()` shall block until a message arrives. If no messages are available at the socket and `O_NONBLOCK` is set on the socket's file descriptor, `recv()` shall fail and set *errno* to `[EAGAIN]` or `[EWOULDBLOCK]`.

RETURN VALUE

Upon successful completion, `recv()` shall return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, `recv()` shall return 0. Otherwise, `-1` shall be returned and *errno* set to indicate the error.

recv()*System Interfaces***56214 ERRORS**

56215 The *recv()* function shall fail if:

56216 [EAGAIN] or [EWOULDBLOCK]

56217 The socket's file descriptor is marked O_NONBLOCK and no data is waiting
56218 to be received; or MSG_OOB is set and no out-of-band data is available and
56219 either the socket's file descriptor is marked O_NONBLOCK or the socket does
56220 not support blocking to await out-of-band data.

56221 [EBADF] The *socket* argument is not a valid file descriptor.

56222 [ECONNRESET] A connection was forcibly closed by a peer.

56223 [EINTR] The *recv()* function was interrupted by a signal that was caught, before any
56224 data was available.

56225 [EINVAL] The MSG_OOB flag is set and no out-of-band data is available.

56226 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

56227 [ENOTSOCK] The *socket* argument does not refer to a socket.

56228 [EOPNOTSUPP] The specified flags are not supported for this socket type or protocol.

56229 [ETIMEDOUT] The connection timed out during connection establishment, or due to a
56230 transmission timeout on active connection.

56231 The *recv()* function may fail if:

56232 [EIO] An I/O error occurred while reading from or writing to the file system.

56233 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

56234 [ENOMEM] Insufficient memory was available to fulfill the request.

56235 EXAMPLES

56236 None.

56237 APPLICATION USAGE

56238 The *recv()* function is equivalent to *recvfrom()* with a zero *address_len* argument, and to *read()* if
56239 no flags are used.

56240 The *select()* and *poll()* functions can be used to determine when data is available to be received.

56241 RATIONALE

56242 None.

56243 FUTURE DIRECTIONS

56244 None.

56245 SEE ALSO

56246 *poll()*, *pselect()*, *read()*, *recvmsg()*, *recvfrom()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*,
56247 *write()*

56248 XBD <sys/socket.h>

56249 CHANGE HISTORY

56250 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

System Interfaces

recvfrom()**NAME**

recvfrom — receive a message from a socket

SYNOPSIS

```
#include <sys/socket.h>

ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
                 int flags, struct sockaddr *restrict address,
                 socklen_t *restrict address_len);
```

DESCRIPTION

The *recvfrom()* function shall receive a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.

The *recvfrom()* function takes the following arguments:

<i>socket</i>	Specifies the socket file descriptor.
<i>buffer</i>	Points to the buffer where the message should be stored.
<i>length</i>	Specifies the length in bytes of the buffer pointed to by the <i>buffer</i> argument.
<i>flags</i>	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:

MSG_PEEK	Peeks at an incoming message. The data is treated as unread and the next <i>recvfrom()</i> or similar function shall still return this data.

MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.

MSG_WAITALL	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.

<i>address</i>	A null pointer, or points to a sockaddr structure in which the sending address is to be stored. The length and format of the address depend on the address family of the socket.

<i>address_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>address</i> argument.

The *recvfrom()* function shall return the length of the message written to the buffer pointed to by the *buffer* argument. For message-based sockets, such as **SOCK_RAW**, **SOCK_DGRAM**, and **SOCK_SEQPACKET**, the entire message shall be read in a single operation. If a message is too long to fit in the supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess bytes shall be discarded. For stream-based sockets, such as **SOCK_STREAM**, message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes available, and no data shall be discarded.

If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first message.

Not all protocols provide the source address for messages. If the *address* argument is not a null pointer and the protocol provides the source address of messages, the source address of the

recvfrom()*System Interfaces*

56295 received message shall be stored in the **sockaddr** structure pointed to by the *address* argument,
 56296 and the length of this address shall be stored in the object pointed to by the *address_len*
 56297 argument.

56298 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
 56299 the stored address shall be truncated.

56300 If the *address* argument is not a null pointer and the protocol does not provide the source address
 56301 of messages, the value stored in the object pointed to by *address* is unspecified.

56302 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
 56303 descriptor, *recvfrom()* shall block until a message arrives. If no messages are available at the
 56304 socket and O_NONBLOCK is set on the socket's file descriptor, *recvfrom()* shall fail and set *errno*
 56305 to [EAGAIN] or [EWOULDBLOCK].

RETURN VALUE

56306 Upon successful completion, *recvfrom()* shall return the length of the message in bytes. If no
 56307 messages are available to be received and the peer has performed an orderly shutdown,
 56308 *recvfrom()* shall return 0. Otherwise, the function shall return -1 and set *errno* to indicate the
 56309 error.
 56310

ERRORS

56311 The *recvfrom()* function shall fail if:

56312 [EAGAIN] or [EWOULDBLOCK]

56313 The socket's file descriptor is marked O_NONBLOCK and no data is waiting
 56314 to be received; or MSG_OOB is set and no out-of-band data is available and
 56315 either the socket's file descriptor is marked O_NONBLOCK or the socket does
 56316 not support blocking to await out-of-band data.
 56317

56318 [EBADF] The *socket* argument is not a valid file descriptor.

56319 [ECONNRESET] A connection was forcibly closed by a peer.

56320 [EINTR] A signal interrupted *recvfrom()* before any data was available.

56321 [EINVAL] The MSG_OOB flag is set and no out-of-band data is available.

56322 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

56323 [ENOTSOCK] The *socket* argument does not refer to a socket.

56324 [EOPNOTSUPP] The specified flags are not supported for this socket type.

56325 [ETIMEDOUT] The connection timed out during connection establishment, or due to a
 56326 transmission timeout on active connection.

56327 The *recvfrom()* function may fail if:

56328 [EIO] An I/O error occurred while reading from or writing to the file system.

56329 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

56330 [ENOMEM] Insufficient memory was available to fulfill the request.

56331 **EXAMPLES**

56332 None.

56333 **APPLICATION USAGE**56334 The *select()* and *poll()* functions can be used to determine when data is available to be received.56335 **RATIONALE**

56336 None.

56337 **FUTURE DIRECTIONS**

56338 None.

56339 **SEE ALSO**56340 *poll()*, *pselect()*, *read()*, *recv()*, *recvmsg()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*, *write()*

56341 XBD <sys/socket.h>

56342 **CHANGE HISTORY**

56343 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

recvmsg()*System Interfaces*56344 **NAME**

56345 recvmsg — receive a message from a socket

56346 **SYNOPSIS**

56347 #include <sys/socket.h>

56348 ssize_t recvmsg(int socket, struct msghdr *message, int flags);

56349 **DESCRIPTION**

56350 The *recvmsg()* function shall receive a message from a connection-mode or connectionless-mode
 56351 socket. It is normally used with connectionless-mode sockets because it permits the application
 56352 to retrieve the source address of received data.

56353 The *recvmsg()* function takes the following arguments:

56354	<i>socket</i>	Specifies the socket file descriptor.
56355	<i>message</i>	Points to a msghdr structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored on input, but may contain meaningful values on output.
56356		
56357		
56358		
56359	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:
56360		
56361	MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
56362		
56363	MSG_PEEK	Peeks at the incoming message.
56364	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.
56365		
56366		
56367		
56368		
56369		

56370 The *recvmsg()* function shall receive messages from unconnected or connected sockets and shall
 56371 return the length of the message.

56372 The *recvmsg()* function shall return the total length of the message. For message-based sockets,
 56373 such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message shall be read in a single
 56374 operation. If a message is too long to fit in the supplied buffers, and MSG_PEEK is not set in the
 56375 *flags* argument, the excess bytes shall be discarded, and MSG_TRUNC shall be set in the
 56376 *msg_flags* member of the **msghdr** structure. For stream-based sockets, such as SOCK_STREAM,
 56377 message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it
 56378 becomes available, and no data shall be discarded.

56379 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
 56380 message.

56381 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
 56382 descriptor, *recvmsg()* shall block until a message arrives. If no messages are available at the
 56383 socket and O_NONBLOCK is set on the socket's file descriptor, the *recvmsg()* function shall fail
 56384 and set *errno* to [EAGAIN] or [EWOULDBLOCK].

56385 In the **msghdr** structure, the *msg_name* and *msg_namelen* members specify the source address if
 56386 the socket is unconnected. If the socket is connected, the *msg_name* and *msg_namelen* members
 56387 shall be ignored. The *msg_name* member may be a null pointer if no names are desired or
 56388 required. The *msg_iov* and *msg_iovlen* fields are used to specify where the received data shall be

stored. *msg_iov* points to an array of **iovec** structures; *msg_iovlen* shall be set to the dimension of this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field gives its size in bytes. Each storage area indicated by *msg_iov* is filled with received data in turn until all of the received data is stored or all of the areas have been filled.

Upon successful completion, the *msg_flags* member of the message header shall be the bitwise-inclusive OR of all of the following flags that indicate conditions detected for the received message:

- MSG_EOR** End-of-record was received (if supported by the protocol).
- MSG_OOB** Out-of-band data was received.
- MSG_TRUNC** Normal data was truncated.
- MSG_CTRUNC** Control data was truncated.

RETURN VALUE

Upon successful completion, *recvmsg()* shall return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, *recvmsg()* shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *recvmsg()* function shall fail if:

[EAGAIN] or [EWOULDBLOCK]

The socket's file descriptor is marked **O_NONBLOCK** and no data is waiting to be received; or **MSG_OOB** is set and no out-of-band data is available and either the socket's file descriptor is marked **O_NONBLOCK** or the socket does not support blocking to await out-of-band data.

[EBADF] The *socket* argument is not a valid open file descriptor.

[ECONNRESET] A connection was forcibly closed by a peer.

[EINTR] This function was interrupted by a signal before any data was available.

[EINVAL] The sum of the *iov_len* values overflows a **ssize_t**, or the **MSG_OOB** flag is set and no out-of-band data is available.

[EMSGSIZE] The *msg_iovlen* member of the **msghdr** structure pointed to by *message* is less than or equal to 0, or is greater than **{IOV_MAX}**.

[ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

[ENOTSOCK] The *socket* argument does not refer to a socket.

[EOPNOTSUPP] The specified flags are not supported for this socket type.

[ETIMEDOUT] The connection timed out during connection establishment, or due to a transmission timeout on active connection.

The *recvmsg()* function may fail if:

[EIO] An I/O error occurred while reading from or writing to the file system.

[ENOBUFS] Insufficient resources were available in the system to perform the operation.

[ENOMEM] Insufficient memory was available to fulfill the request.

recvmsg()*System Interfaces*56427 **EXAMPLES**

56428 None.

56429 **APPLICATION USAGE**56430 The *select()* and *poll()* functions can be used to determine when data is available to be received.56431 **RATIONALE**

56432 None.

56433 **FUTURE DIRECTIONS**

56434 None.

56435 **SEE ALSO**56436 *poll()*, *pselect()*, *recv()*, *recvfrom()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*56437 XBD <[sys/socket.h](#)>56438 **CHANGE HISTORY**

56439 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

regcomp, regerror, regex, regfree — regular expression matching

SYNOPSIS

```
#include <regex.h>

int regcomp(regex_t *restrict preg, const char *restrict pattern,
            int cflags);
size_t regerror(int errcode, const regex_t *restrict preg,
               char *restrict errbuf, size_t errbuf_size);
int regex(const regex_t *restrict preg, const char *restrict string,
          size_t nmatch, regmatch_t pmatch[restrict], int eflags);
void regfree(regex_t *preg);
```

DESCRIPTION

These functions interpret *basic* and *extended* regular expressions as described in XBD [Chapter 9](#) (on page 181).

The **regex_t** structure is defined in **<regex.h>** and contains at least the following member:

Member Type	Member Name	Description
size_t	<i>re_nsub</i>	Number of parenthesized subexpressions.

The **regmatch_t** structure is defined in **<regex.h>** and contains at least the following members:

Member Type	Member Name	Description
regoff_t	<i>rm_so</i>	Byte offset from start of <i>string</i> to start of substring.
regoff_t	<i>rm_eo</i>	Byte offset from start of <i>string</i> of the first character after the end of substring.

The *regcomp()* function shall compile the regular expression contained in the string pointed to by the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in the **<regex.h>** header:

REG_EXTENDED Use Extended Regular Expressions.

REG_ICASE Ignore case in match (see XBD [Chapter 9](#), on page 181).

REG_NOSUB Report only success/fail in *regex*.

REG_NEWLINE Change the handling of <newline> characters, as described in the text.

The default regular expression type for *pattern* is a Basic Regular Expression. The application can specify Extended Regular Expressions using the **REG_EXTENDED** *cflags* flag.

If the **REG_NOSUB** flag was not set in *cflags*, then *regcomp()* shall set *re_nsub* to the number of parenthesized subexpressions (delimited by "`\( \)`" in basic regular expressions or "`( )`" in extended regular expressions) found in *pattern*.

The *regex()* function compares the null-terminated string specified by *string* with the compiled regular expression *preg* initialized by a previous call to *regcomp()*. If it finds a match, *regex()* shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in the **<regex.h>** header:

regcomp()*System Interfaces*

56482	REG_NOTBOL	The first character of the string pointed to by <i>string</i> is not the beginning of the line. Therefore, the <circumflex> character ('^'), when taken as a special character, shall not match the beginning of <i>string</i> .
56483		
56484		
56485	REG_NOTEOL	The last character of the string pointed to by <i>string</i> is not the end of the line. Therefore, the <dollar-sign> ('\$'), when taken as a special character, shall not match the end of <i>string</i> .
56486		
56487		
56488		If <i>nmatch</i> is 0 or REG_NOSUB was set in the <i>cflags</i> argument to <i>regcomp()</i> , then <i>regexexec()</i> shall ignore the <i>pmatch</i> argument. Otherwise, the application shall ensure that the <i>pmatch</i> argument points to an array with at least <i>nmatch</i> elements, and <i>regexexec()</i> shall fill in the elements of that array with offsets of the substrings of <i>string</i> that correspond to the parenthesized subexpressions of <i>pattern</i> : <i>pmatch[i].rm_so</i> shall be the byte offset of the beginning and <i>pmatch[i].rm_eo</i> shall be one greater than the byte offset of the end of substring <i>i</i> . (Subexpression <i>i</i> begins at the <i>i</i> th matched open parenthesis, counting from 1.) Offsets in <i>pmatch</i> [0] identify the substring that corresponds to the entire regular expression. Unused elements of <i>pmatch</i> up to <i>pmatch</i> [<i>nmatch</i> −1] shall be filled with −1. If there are more than <i>nmatch</i> subexpressions in <i>pattern</i> (<i>pattern</i> itself counts as a subexpression), then <i>regexexec()</i> shall still do the match, but shall record only the first <i>nmatch</i> substrings.
56489		
56490		
56491		
56492		
56493		
56494		
56495		
56496		
56497		
56498		
56499		When matching a basic or extended regular expression, any given parenthesized subexpression of <i>pattern</i> might participate in the match of several different substrings of <i>string</i> , or it might not match any substring even though the pattern as a whole did match. The following rules shall be used to determine which substrings to report in <i>pmatch</i> when matching regular expressions:
56500		
56501		
56502		
56503		1. If subexpression <i>i</i> in a regular expression is not contained within another subexpression, and it participated in the match several times, then the byte offsets in <i>pmatch</i> [<i>i</i>] shall delimit the last such match.
56504		
56505		
56506		2. If subexpression <i>i</i> is not contained within another subexpression, and it did not participate in an otherwise successful match, the byte offsets in <i>pmatch</i> [<i>i</i>] shall be −1. A subexpression does not participate in the match when:
56507		
56508		
56509		'*' or "\{\}" appears immediately after the subexpression in a basic regular
56510		expression, or '.*', '?', or "{ }" appears immediately after the subexpression in
56511		an extended regular expression, and the subexpression did not match (matched 0
56512		times)
56513	or:	
56514		' ' is used in an extended regular expression to select this subexpression or
56515		another, and the other subexpression matched.
56516		3. If subexpression <i>i</i> is contained within another subexpression <i>j</i> , and <i>i</i> is not contained
56517		within any other subexpression that is contained within <i>j</i> , and a match of subexpression <i>j</i>
56518		is reported in <i>pmatch</i> [<i>j</i>], then the match or non-match of subexpression <i>i</i> reported in
56519		<i>pmatch</i> [<i>i</i>] shall be as described in 1. and 2. above, but within the substring reported in
56520		<i>pmatch</i> [<i>j</i>] rather than the whole string. The offsets in <i>pmatch</i> [<i>i</i>] are still relative to the start
56521		of <i>string</i> .
56522		4. If subexpression <i>i</i> is contained in subexpression <i>j</i> , and the byte offsets in <i>pmatch</i> [<i>j</i>] are −1,
56523		then the pointers in <i>pmatch</i> [<i>i</i>] shall also be −1.
56524		5. If subexpression <i>i</i> matched a zero-length string, then both byte offsets in <i>pmatch</i> [<i>i</i>] shall be
56525		the byte offset of the character or null terminator immediately following the zero-length
56526		string.

If, when *regexec()* is called, the locale is different from when the regular expression was compiled, the result is undefined.

If REG_NEWLINE is not set in *cflags*, then a <newline> in *pattern* or *string* shall be treated as an ordinary character. If REG_NEWLINE is set, then <newline> shall be treated as an ordinary character except as follows:

1. A <newline> in *string* shall not be matched by a <period> outside a bracket expression or by any form of a non-matching list (see XBD Chapter 9, on page 181).
2. A <circumflex> (' ^ ') in *pattern*, when used to specify expression anchoring (see XBD Section 9.3.8, on page 187), shall match the zero-length string immediately after a <newline> in *string*, regardless of the setting of REG_NOTBOL.
3. A <dollar-sign> (' \$ ') in *pattern*, when used to specify expression anchoring, shall match the zero-length string immediately before a <newline> in *string*, regardless of the setting of REG_NOTEOL.

The *regfree()* function frees any memory allocated by *regcomp()* associated with *preg*.

The following constants are defined as error return values:

REG_BADBR	Content of "\{\}" invalid: not a number, number too large, more than two numbers, first larger than second.
REG_BADPAT	Invalid regular expression.
REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.
REG_EBRACE	"\{\}" imbalance.
REG_EBRACK	"[]" imbalance.
REG_ECOLLATE	Invalid collating element referenced.
REG_ECTYPE	Invalid character class type referenced.
REG_EESCAPE	Trailing <backslash> character in pattern.
REG_EPAREN	"\(" or ")" imbalance.
REG_ERANGE	Invalid endpoint in range expression.
REG_ESPACE	Out of memory.
REG_ESUBREG	Number in "\digit" invalid or in error.
REG_NOMATCH	<i>regexec()</i> failed to match.

If more than one error occurs in processing a function call, any one of the possible constants may be returned, as the order of detection is unspecified.

The *regerror()* function provides a mapping from error codes returned by *regcomp()* and *regexec()* to unspecified printable strings. It generates a string corresponding to the value of the *errcode* argument, which the application shall ensure is the last non-zero value returned by *regcomp()* or *regexec()* with the given value of *preg*. If *errcode* is not such a value, the content of the generated string is unspecified.

If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexec()* or *regcomp()*, the *regerror()* still generates an error string corresponding to the value of *errcode*, but it might not be as detailed under some implementations.

If the *errbuf_size* argument is not 0, *regerror()* shall place the generated string into the buffer of

regcomp()*System Interfaces*

size *errbuf_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit in the buffer, *regerror()* shall truncate the string and null-terminate the result.

If *errbuf_size* is 0, *regerror()* shall ignore the *errbuf* argument, and return the size of the buffer needed to hold the generated string.

If the *preg* argument to *regexexec()* or *regfree()* is not a compiled regular expression returned by *regcomp()*, the result is undefined. A *preg* is no longer treated as a compiled regular expression after it is given to *regfree()*.

RETURN VALUE

Upon successful completion, the *regcomp()* function shall return 0. Otherwise, it shall return an integer value indicating an error as described in **<regex.h>**, and the content of *preg* is undefined. If a code is returned, the interpretation shall be as given in **<regex.h>**.

If *regcomp()* detects an invalid RE, it may return REG_BADPAT, or it may return one of the error codes that more precisely describes the error.

Upon successful completion, the *regexexec()* function shall return 0. Otherwise, it shall return REG_NOMATCH to indicate no match.

Upon successful completion, the *regerror()* function shall return the number of bytes needed to hold the entire generated string, including the null termination. If the return value is greater than *errbuf_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

The *regfree()* function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

```
#include <regex.h>

/*
 * Match string against the extended regular expression in
 * pattern, treating errors as no match.
 *
 * Return 1 for match, 0 for no match.
 */

int
match(const char *string, char *pattern)
{
    int    status;
    regex_t re;

    if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
        return(0);        /* Report error. */
    }
    status = regexexec(&re, string, (size_t) 0, NULL, 0);
    regfree(&re);
    if (status != 0) {
        return(0);        /* Report error. */
    }
    return(1);
}
```

The following demonstrates how the REG_NOTBOL flag could be used with *regexexec()* to find all

substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very little error checking is done.)

```
(void) regcomp (&re, pattern, 0);
/* This call to regexexec() finds the first match on the line. */
error = regexexec (&re, &buffer[0], 1, &pm, 0);
while (error == 0) { /* While matches found. */
    /* Substring found between pm.rm_so and pm.rm_eo. */
    /* This call to regexexec() finds the next match. */
    error = regexexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
}
```

APPLICATION USAGE

An application could use:

```
regerror(code, preg, (char *)NULL, (size_t)0)
```

to find out how big a buffer is needed for the generated string, *malloc()* a buffer to hold the string, and then call *regerror()* again to get the string. Alternatively, it could allocate a fixed, static buffer that is big enough to hold most strings, and then use *malloc()* to allocate a larger buffer if it finds that this is too small.

To match a pattern as described in XCU [Section 2.13](#) (on page 2332), use the *fnmatch()* function.

RATIONALE

The *regexexec()* function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are supplied by the application, even if some elements of *pmatch* do not correspond to subexpressions in *pattern*. The application developer should note that there is probably no reason for using a value of *nmatch* that is larger than *preg->re_nsub*+1.

The REG_NEWLINE flag supports a use of RE matching that is needed in some applications like text editors. In such applications, the user supplies an RE asking the application to find a line that matches the given expression. An anchor in such an RE anchors at the beginning or end of any line. Such an application can pass a sequence of <newline>-separated lines to *regexexec()* as a single long string and specify REG_NEWLINE to *regcomp()* to get the desired behavior. The application must ensure that there are no explicit <newline> characters in *pattern* if it wants to ensure that any match occurs entirely within a single line.

The REG_NEWLINE flag affects the behavior of *regexexec()*, but it is in the *cflags* parameter to *regcomp()* to allow flexibility of implementation. Some implementations will want to generate the same compiled RE in *regcomp()* regardless of the setting of REG_NEWLINE and have *regexexec()* handle anchors differently based on the setting of the flag. Other implementations will generate different compiled REs based on the REG_NEWLINE.

The REG_ICASE flag supports the operations taken by the *grep -i* option and the historical implementations of *ex* and *vi*. Including this flag will make it easier for application code to be written that does the same thing as these utilities.

The substrings reported in *pmatch[]* are defined using offsets from the start of the string rather than pointers. This allows type-safe access to both constant and non-constant strings.

The type **regoff_t** is used for the elements of *pmatch[]* to ensure that the application can represent large arrays in memory (important for an application conforming to the Shell and Utilities volume of POSIX.1-2008).

The 1992 edition of this standard required **regoff_t** to be at least as wide as **off_t**, to facilitate future extensions in which the string to be searched is taken from a file. However, these future extensions have not appeared. The requirement rules out popular implementations with 32-bit

regcomp()*System Interfaces*

56658 **regoff_t** and 64-bit **off_t**, so it has been removed.

56659 The standard developers rejected the inclusion of a *regsub()* function that would be used to do
 56660 substitutions for a matched RE. While such a routine would be useful to some applications, its
 56661 utility would be much more limited than the matching function described here. Both RE parsing
 56662 and substitution are possible to implement without support other than that required by the
 56663 ISO C standard, but matching is much more complex than substituting. The only difficult part of
 56664 substitution, given the information supplied by *regexexec()*, is finding the next character in a string
 56665 when there can be multi-byte characters. That is a much larger issue, and one that needs a more
 56666 general solution.

56667 The *errno* variable has not been used for error returns to avoid filling the *errno* name space for
 56668 this feature.

56669 The interface is defined so that the matched substrings *rm_sp* and *rm_ep* are in a separate
 56670 **regmatch_t** structure instead of in **regex_t**. This allows a single compiled RE to be used
 56671 simultaneously in several contexts; in *main()* and a signal handler, perhaps, or in multiple
 56672 threads of lightweight processes. (The *preg* argument to *regexexec()* is declared with type **const**, so
 56673 the implementation is not permitted to use the structure to store intermediate results.) It also
 56674 allows an application to request an arbitrary number of substrings from an RE. The number of
 56675 subexpressions in the RE is reported in *re_nsub* in *preg*. With this change to *regexexec()*,
 56676 consideration was given to dropping the REG_NOSUB flag since the user can now specify this
 56677 with a zero *nmatch* argument to *regexexec()*. However, keeping REG_NOSUB allows an
 56678 implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp()* that
 56679 no subexpressions need be reported. The implementation is only required to fill in *pmatch* if
 56680 *nmatch* is not zero and if REG_NOSUB is not specified. Note that the **size_t** type, as defined in
 56681 the ISO C standard, is unsigned, so the description of *regexexec()* does not need to address
 56682 negative values of *nmatch*.

56683 REG_NOTBOL was added to allow an application to do repeated searches for the same pattern
 56684 in a line. If the pattern contains a `<circumflex>` character that should match the beginning of a
 56685 line, then the pattern should only match when matched against the beginning of the line.
 56686 Without the REG_NOTBOL flag, the application could rewrite the expression for subsequent
 56687 matches, but in the general case this would require parsing the expression. The need for
 56688 REG_NOTEOL is not as clear; it was added for symmetry.

56689 The addition of the *regerror()* function addresses the historical need for conforming application
 56690 programs to have access to error information more than "Function failed to compile/match your
 56691 RE for unknown reasons".

56692 This interface provides for two different methods of dealing with error conditions. The specific
 56693 error codes (REG_EBRACE, for example), defined in **<regex.h>**, allow an application to recover
 56694 from an error if it is so able. Many applications, especially those that use patterns supplied by a
 56695 user, will not try to deal with specific error cases, but will just use *regerror()* to obtain a human-
 56696 readable error message to present to the user.

56697 The *regerror()* function uses a scheme similar to *confstr()* to deal with the problem of allocating
 56698 memory to hold the generated string. The scheme used by *strerror()* in the ISO C standard was
 56699 considered unacceptable since it creates difficulties for multi-threaded applications.

56700 The *preg* argument is provided to *regerror()* to allow an implementation to generate a more
 56701 descriptive message than would be possible with *errcode* alone. An implementation might, for
 56702 example, save the character offset of the offending character of the pattern in a field of *preg*, and
 56703 then include that in the generated message string. The implementation may also ignore *preg*.

56704 A REG_FILENAME flag was considered, but omitted. This flag caused *regexexec()* to match

patterns as described in XCU Section 2.13 (on page 2332) instead of REs. This service is now provided by the *fnmatch()* function.

Notice that there is a difference in philosophy between the ISO POSIX-2:1993 standard and POSIX.1-2008 in how to handle a “bad” regular expression. The ISO POSIX-2:1993 standard says that many bad constructs “produce undefined results”, or that “the interpretation is undefined”. POSIX.1-2008, however, says that the interpretation of such REs is unspecified. The term “undefined” means that the action by the application is an error, of similar severity to passing a bad pointer to a function.

The *regcomp()* and *regexexec()* functions are required to accept any null-terminated string as the *pattern* argument. If the meaning of the string is “undefined”, the behavior of the function is “unspecified”. POSIX.1-2008 does not specify how the functions will interpret the pattern; they might return error codes, or they might do pattern matching in some completely unexpected way, but they should not do something like abort the process.

FUTURE DIRECTIONS

None.

SEE ALSO

fnmatch(), *glob()*

XBD Chapter 9 (on page 181), *<regex.h>*, *<sys/types.h>*

XCU Section 2.13 (on page 2332)

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

Moved from POSIX2 C-language Binding to BASE.

Issue 6

In the SYNOPSIS, the optional include of the *<sys/types.h>* header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

The normative text is updated to avoid use of the term “must” for application requirements.

The REG_ENOSYS constant is removed.

The **restrict** keyword is added to the *regcomp()*, *regerror()*, and *regexexec()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #134 is applied, clarifying that if more than one error occurs in processing a function call, any one of the possible constants may be returned.

SD5-XBD-ERN-60 is applied.

remainder()*System Interfaces*56743 **NAME**

56744 remainder, remainderf, remainderl — remainder function

56745 **SYNOPSIS**

56746 #include <math.h>

56747 double remainder(double x, double y);

56748 float remainderf(float x, float y);

56749 long double remainderl(long double x, long double y);

56750 **DESCRIPTION**

56751 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 56752 conflict between the requirements described here and the ISO C standard is unintentional. This
 56753 volume of POSIX.1-2008 defers to the ISO C standard.

56754 These functions shall return the floating-point remainder $r=x-ny$ when y is non-zero. The value
 56755 n is the integral value nearest the exact value x/y . When $|n-x/y|=\frac{1}{2}$, the value n is chosen to
 56756 be even.

56757 The behavior of *remainder()* shall be independent of the rounding mode.

56758 **RETURN VALUE**

56759 Upon successful completion, these functions shall return the floating-point remainder $r=x-ny$
 56760 when y is non-zero.

56761 On systems that do not support the IEC 60559 Floating-Point option, if y is zero, it is
 56762 implementation-defined whether a domain error occurs or zero is returned.

56763 MX If x or y is NaN, a NaN shall be returned.

56764 If x is infinite or y is 0 and the other is non-NaN, a domain error shall occur, and either a NaN (if
 56765 supported), or an implementation-defined value shall be returned.

56766 **ERRORS**

56767 These functions shall fail if:

56768 MX **Domain Error** The x argument is $\pm\text{Inf}$, or the y argument is ± 0 and the other argument is non-
 56769 NaN.

56770 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 56771 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 56772 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 56773 shall be raised.

56774 These functions may fail if:

56775 **Domain Error** The y argument is zero.

56776 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 56777 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 56778 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 56779 shall be raised.

56780 **EXAMPLES**

56781 None.

56782 **APPLICATION USAGE**

56783 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 56784 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

56785 **RATIONALE**

56786 None.

56787 **FUTURE DIRECTIONS**

56788 None.

56789 **SEE ALSO**56790 *abs()*, *div()*, *feclearexcept()*, *fetestexcept()*, *ldiv()*

56791 XBD Section 4.19 (on page 116), <math.h>

56792 **CHANGE HISTORY**

56793 First released in Issue 4, Version 2.

56794 **Issue 5**

56795 Moved from X/OPEN UNIX extension to BASE.

56796 **Issue 6**56797 The *remainder()* function is no longer marked as an extension.

56798 The *remainderf()* and *remainderl()* functions are added for alignment with the ISO/IEC 9899:1999
 56799 standard.

56800 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 56801 revised to align with the ISO/IEC 9899:1999 standard.

56802 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 56803 marked.

56804 **Issue 7**

56805 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #55 (SD5-XSH-ERN-82) is applied.

remove()

System Interfaces

56806 **NAME**56807 **remove** — remove a file56808 **SYNOPSIS**56809 `#include <stdio.h>`56810 `int remove(const char *path);`56811 **DESCRIPTION**

56812 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 56813 conflict between the requirements described here and the ISO C standard is unintentional. This
 56814 volume of POSIX.1-2008 defers to the ISO C standard.

56815 The *remove()* function shall cause the file named by the pathname pointed to by *path* to be no
 56816 longer accessible by that name. A subsequent attempt to open that file using that name shall fail,
 56817 unless it is created anew.

56818 CX If *path* does not name a directory, *remove(path)* shall be equivalent to *unlink(path)*.

56819 If *path* names a directory, *remove(path)* shall be equivalent to *rmdir(path)*.

56820 **RETURN VALUE**

56821 CX Refer to *rmdir()* or *unlink()*.

56822 **ERRORS**

56823 CX Refer to *rmdir()* or *unlink()*.

56824 **EXAMPLES**56825 **Removing Access to a File**

56826 The following example shows how to remove access to a file named */home/cnd/old_mods*.

```
56827 #include <stdio.h>
56828 int status;
56829 ...
56830 status = remove("/home/cnd/old_mods");
```

56831 **APPLICATION USAGE**

56832 None.

56833 **RATIONALE**

56834 None.

56835 **FUTURE DIRECTIONS**

56836 None.

56837 **SEE ALSO**

56838 *rmdir()*, *unlink()*

56839 XBD *<stdio.h>*

56840 **CHANGE HISTORY**

56841 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ISO C
 56842 standard.

56843 **Issue 6**

56844 Extensions beyond the ISO C standard are marked.

- 56845 The following new requirements on POSIX implementations derive from alignment with the
56846 Single UNIX Specification:
- 56847 • The DESCRIPTION, RETURN VALUE, and ERRORS sections are updated so that if *path* is
56848 not a directory, *remove()* is equivalent to *unlink()*, and if it is a directory, it is equivalent to
56849 *rmdir()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

remque()*System Interfaces*56850 **NAME**

56851 remque — remove an element from a queue

56852 **SYNOPSIS**56853 XSI `#include <search.h>`56854 `void remque(void *element);`56855 **DESCRIPTION**56856 Refer to *insque()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

remquo, remquof, remquol — remainder functions

SYNOPSIS

```
#include <math.h>

double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
long double remquol(long double x, long double y, int *quo);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *remquo()*, *remquof()*, and *remquol()* functions shall compute the same remainder as the *remainder()*, *remainderf()*, and *remainderl()* functions, respectively. In the object pointed to by *quo*, they store a value whose sign is the sign of x/y and whose magnitude is congruent modulo 2^n to the magnitude of the integral quotient of x/y , where n is an implementation-defined integer greater than or equal to 3. If y is zero, the value stored in the object pointed to by *quo* is unspecified.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

These functions shall return $x \text{ REM } y$.

On systems that do not support the IEC 60559 Floating-Point option, if y is zero, it is implementation-defined whether a domain error occurs or zero is returned.

MX If x or y is NaN, a NaN shall be returned.

If x is $\pm\text{Inf}$ or y is zero and the other argument is non-NaN, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

ERRORS

These functions shall fail if:

MX **Domain Error** The x argument is $\pm\text{Inf}$, or the y argument is ± 0 and the other argument is non-NaN.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error The y argument is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

remquo()

System Interfaces

56899 **EXAMPLES**

56900 None.

56901 **APPLICATION USAGE**

56902 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
56903 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

56904 **RATIONALE**

56905 These functions are intended for implementing argument reductions which can exploit a few
56906 low-order bits of the quotient. Note that *x* may be so large in magnitude relative to *y* that an
56907 exact representation of the quotient is not practical.

56908 **FUTURE DIRECTIONS**

56909 None.

56910 **SEE ALSO**56911 [*feclearexcept\(\)*](#), [*fetestexcept\(\)*](#), [*remainder\(\)*](#)56912 XBD [Section 4.19](#) (on page 116), [**<math.h>**](#)56913 **CHANGE HISTORY**

56914 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

56915 **Issue 7**

56916 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #56 (SD5-XSH-ERN-83) is applied.

System Interfaces

rename()56917 **NAME**56918 `rename, renameat` — rename file relative to directory file descriptor56919 **SYNOPSIS**56920 `#include <stdio.h>`56921 `int rename(const char *old, const char *new);`56922 CX `int renameat(int oldfd, const char *old, int newfd,`
56923 `const char *new);`56924 **DESCRIPTION**56925 CX For `rename()`: The functionality described on this reference page is aligned with the ISO C
56926 standard. Any conflict between the requirements described here and the ISO C standard is
56927 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.56928 The `rename()` function shall change the name of a file. The `old` argument points to the pathname
56929 CX of the file to be renamed. The `new` argument points to the new pathname of the file. If the `new`
56930 argument does not resolve to an existing directory entry for a file of type directory and the `new`
56931 argument contains at least one non-`<slash>` character and ends with one or more trailing
56932 `<slash>` characters after all symbolic links have been processed, `rename()` shall fail.56933 If either the `old` or `new` argument names a symbolic link, `rename()` shall operate on the symbolic
56934 link itself, and shall not resolve the last component of the argument. If the `old` argument and the
56935 `new` argument resolve to either the same existing directory entry or different directory entries for
56936 the same existing file, `rename()` shall return successfully and perform no other action.56937 If the `old` argument points to the pathname of a file that is not a directory, the `new` argument shall
56938 not point to the pathname of a directory. If the link named by the `new` argument exists, it shall be
56939 removed and `old` renamed to `new`. In this case, a link named `new` shall remain visible to other
56940 processes throughout the renaming operation and refer either to the file referred to by `new` or `old`
56941 before the operation began. Write access permission is required for both the directory containing
56942 `old` and the directory containing `new`.56943 If the `old` argument points to the pathname of a directory, the `new` argument shall not point to the
56944 pathname of a file that is not a directory. If the directory named by the `new` argument exists, it
56945 shall be removed and `old` renamed to `new`. In this case, a link named `new` shall exist throughout
56946 the renaming operation and shall refer either to the directory referred to by `new` or `old` before the
56947 operation began. If `new` names an existing directory, it shall be required to be an empty directory.56948 If either `pathname` argument refers to a path whose final component is either dot or dot-dot,
56949 `rename()` shall fail.56950 If the `old` argument points to a pathname of a symbolic link, the symbolic link shall be renamed.
56951 If the `new` argument points to a pathname of a symbolic link, the symbolic link shall be removed.56952 The `old` pathname shall not name an ancestor directory of the `new` pathname. Write access
56953 permission is required for the directory containing `old` and the directory containing `new`. If the
56954 `old` argument points to the pathname of a directory, write access permission may be required for
56955 the directory named by `old`, and, if it exists, the directory named by `new`.56956 If the link named by the `new` argument exists and the file's link count becomes 0 when it is
56957 removed and no process has the file open, the space occupied by the file shall be freed and the
56958 file shall no longer be accessible. If one or more processes have the file open when the last link is
56959 removed, the link shall be removed before `rename()` returns, but the removal of the file contents
56960 shall be postponed until all references to the file are closed.56961 Upon successful completion, `rename()` shall mark for update the last data modification and last

rename()*System Interfaces*

file status change timestamps of the parent directory of each file.

If the *rename()* function fails for any reason other than [EIO], any file named by *new* shall be unaffected.

The *renameat()* function shall be equivalent to the *rename()* function except in the case where either *old* or *new* specifies a relative path. If *old* is a relative path, the file to be renamed is located relative to the directory associated with the file descriptor *oldfd* instead of the current working directory. If *new* is a relative path, the same happens only relative to the directory associated with *newfd*. If the file descriptor was opened without O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function shall not perform the check.

If *renameat()* is passed the special value AT_FDCWD in the *oldfd* or *newfd* parameter, the current working directory shall be used in the determination of the file for the respective *path* parameter.

RETURN VALUE

CX Upon successful completion, the *rename()* function shall return 0. Otherwise, it shall return -1, *errno* shall be set to indicate the error, and neither the file named by *old* nor the file named by *new* shall be changed or created.

CX Upon successful completion, the *renameat()* function shall return 0. Otherwise, it shall return -1 and set *errno* to indicate the error.

ERRORS

CX The *rename()* and *renameat()* functions shall fail if:

CX [EACCES] A component of either path prefix denies search permission; or one of the directories containing *old* or *new* denies write permissions; or, write permission is required and is denied for a directory pointed to by the *old* or *new* arguments.

CX [EBUSY] The directory named by *old* or *new* is currently in use by the system or another process, and the implementation considers this an error.

CX [EEXIST] or [ENOTEMPTY] The link named by *new* is a directory that is not an empty directory.

CX [EINVAL] The *old* pathname names an ancestor directory of the *new* pathname, or either *pathname* argument contains a final component that is dot or dot-dot.

CX [EIO] A physical I/O error has occurred.

CX [EISDIR] The *new* argument points to a directory and the *old* argument points to a file that is not a directory.

CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

CX [EMLINK] The file named by *old* is a directory, and the link count of the parent directory of *new* would exceed {LINK_MAX}.

CX [ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

CX [ENOENT] The link named by *old* does not name an existing file, a component of the path prefix of *new* does not exist, or either *old* or *new* points to an empty string.

System Interfaces

rename()

57004	CX	[ENOSPC]	The directory that would contain <i>new</i> cannot be extended.
57005	CX	[ENOTDIR]	A component of either path prefix is not a directory; or the <i>old</i> argument names a directory and the <i>new</i> argument names a non-directory file; or the <i>old</i> argument contains at least one non- <code><slash></code> character and ends with one or more trailing <code><slash></code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory; or the <i>new</i> argument names a nonexistent file, contains at least one non- <code><slash></code> character, and ends with one or more trailing <code><slash></code> characters.
57012	XSI	[EPERM] or [EACCES]	The <code>S_ISVTX</code> flag is set on the directory containing the file referred to by <i>old</i> and the process does not satisfy the criteria specified in XBD Section 4.2 (on page 107) with respect to <i>old</i> ; or <i>new</i> refers to an existing file, the <code>S_ISVTX</code> flag is set on the directory containing this file, and the process does not satisfy the criteria specified in XBD Section 4.2 with respect to this file.
57018	CX	[EROFS]	The requested operation requires writing in a directory on a read-only file system.
57020	CX	[EXDEV]	The links named by <i>new</i> and <i>old</i> are on different file systems and the implementation does not support links between file systems.
57022	CX	In addition, the <i>renameat()</i> function shall fail if:	
57023		[EACCES]	<i>oldfd</i> or <i>newfd</i> was not opened with <code>O_SEARCH</code> and the permissions of the directory underlying <i>oldfd</i> or <i>newfd</i> respectively do not permit directory searches.
57026		[EBADF]	The <i>old</i> argument does not specify an absolute path and the <i>oldfd</i> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching, or the <i>new</i> argument does not specify an absolute path and the <i>newfd</i> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching.
57031	CX	The <i>rename()</i> and <i>renameat()</i> functions may fail if:	
57032	OB XSR	[EBUSY]	The file named by the <i>old</i> or <i>new</i> arguments is a named STREAM.
57033	CX	[ELOOP]	More than <code>{SYMLOOP_MAX}</code> symbolic links were encountered during resolution of the <i>path</i> argument.
57035	CX	[ENAMETOOLONG]	The length of a pathname exceeds <code>{PATH_MAX}</code> , or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds <code>{PATH_MAX}</code> .
57039	CX	[ETXTBSY]	The file to be renamed is a pure procedure (shared text) file that is being executed.
57041	CX	The <i>renameat()</i> function may fail if:	
57042		[ENOTDIR]	The <i>old</i> argument is not an absolute path and <i>oldfd</i> is neither <code>AT_FDCWD</code> nor a file descriptor associated with a directory, or the <i>new</i> argument is not an absolute path and <i>newfd</i> is neither <code>AT_FDCWD</code> nor a file descriptor associated with a directory.

rename()*System Interfaces*57046 **EXAMPLES**57047 **Renaming a File**

57048 The following example shows how to rename a file named `/home/cnd/mod1` to
 57049 `/home/cnd/mod2`.

```
57050 #include <stdio.h>
57051 int status;
57052 ...
57053 status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

57054 **APPLICATION USAGE**

57055 Some implementations mark for update the last file status change timestamp of renamed files
 57056 and some do not. Applications which make use of the last file status change timestamp may
 57057 behave differently with respect to renamed files unless they are designed to allow for either
 57058 behavior.

57059 **RATIONALE**

57060 This *rename()* function is equivalent for regular files to that defined by the ISO C standard. Its
 57061 inclusion here expands that definition to include actions on directories and specifies behavior
 57062 when the *new* parameter names a file that already exists. That specification requires that the
 57063 action of the function be atomic.

57064 One of the reasons for introducing this function was to have a means of renaming directories
 57065 while permitting implementations to prohibit the use of *link()* and *unlink()* with directories,
 57066 thus constraining links to directories to those made by *mkdir()*.

57067 The specification that if *old* and *new* refer to the same file is intended to guarantee that:

```
57068 rename("x", "x");
```

57069 does not remove the file.

57070 Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

57071 See also the descriptions of [ENOTEMPTY] and [ENAMETOOLONG] in *rmdir()* and [EBUSY] in
 57072 *unlink()*. For a discussion of [EXDEV], see *link()*.

57073 The purpose of the *renameat()* function is to rename files in directories other than the current
 57074 working directory without exposure to race conditions. Any part of the path of a file could be
 57075 changed in parallel to a call to *rename()*, resulting in unspecified behavior. By opening file
 57076 descriptors for the source and target directories and using the *renameat()* function it can be
 57077 guaranteed that that renamed file is located correctly and the resulting file is in the desired
 57078 directory.

57079 **FUTURE DIRECTIONS**

57080 None.

57081 **SEE ALSO**

57082 *link()*, *rmdir()*, *symlink()*, *unlink()*

57083 XBD Section 4.2 (on page 107), `<stdio.h>`

57084 **CHANGE HISTORY**

57085 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The [EBUSY] error is added to the optional part of the ERRORS section.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] mandatory error condition is added.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Details are added regarding the treatment of symbolic links.
- The [ELOOP] optional error condition is added.

The normative text is updated to avoid use of the term “must” for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #016 is applied, changing the definition of the [ENOTDIR] error.

Austin Group Interpretation 1003.1-2001 #076 is applied, clarifying the behavior if the final component of a path is either dot or dot-dot, and adding the associated [EINVAL] error case.

Austin Group Interpretation 1003.1-2001 #143 is applied.

Austin Group Interpretation 1003.1-2001 #145 is applied, clarifying that the [ENOENT] error condition also applies to the case in which a component of *new* does not exist.

Austin Group Interpretations 1003.1-2001 #174 and #181 are applied.

The *renameat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

rewind()57113 **NAME**57114 `rewind` — reset the file position indicator in a stream57115 **SYNOPSIS**57116 `#include <stdio.h>`57117 `void rewind(FILE *stream);`57118 **DESCRIPTION**

57119 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 57120 conflict between the requirements described here and the ISO C standard is unintentional. This
 57121 volume of POSIX.1-2008 defers to the ISO C standard.

57122 The call:

57123 `rewind(stream)`

57124 shall be equivalent to:

57125 `(void) fseek(stream, 0L, SEEK_SET)`57126 except that `rewind()` shall also clear the error indicator.

57127 CX Since `rewind()` does not return a value, an application wishing to detect errors should clear `errno`,
 57128 then call `rewind()`, and if `errno` is non-zero, assume an error has occurred.

57129 **RETURN VALUE**57130 The `rewind()` function shall not return a value.57131 **ERRORS**57132 CX Refer to `fseek()` with the exception of [EINVAL] which does not apply.57133 **EXAMPLES**

57134 None.

57135 **APPLICATION USAGE**

57136 None.

57137 **RATIONALE**

57138 None.

57139 **FUTURE DIRECTIONS**

57140 None.

57141 **SEE ALSO**57142 `fseek()`57143 XBD `<stdio.h>`57144 **CHANGE HISTORY**

57145 First released in Issue 1. Derived from Issue 1 of the SVID.

57146 **Issue 6**

57147 Extensions beyond the ISO C standard are marked.

57148 NAME

57149 **rewinddir** — reset the position of a directory stream to the beginning of a directory

57150 SYNOPSIS

57151 `#include <dirent.h>`
 57152 `void rewinddir(DIR *dirp);`

57153 DESCRIPTION

57154 The *rewinddir()* function shall reset the position of the directory stream to which *dirp* refers to the
 57155 beginning of the directory. It shall also cause the directory stream to refer to the current state of
 57156 the corresponding directory, as a call to *opendir()* would have done. If *dirp* does not refer to a
 57157 directory stream, the effect is undefined.

57158 After a call to the *fork()* function, either the parent or child (but not both) may continue
 57159 XSI processing the directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and
 57160 child processes use these functions, the result is undefined.

57161 RETURN VALUE

57162 The *rewinddir()* function shall not return a value.

57163 ERRORS

57164 No errors are defined.

57165 EXAMPLES

57166 None.

57167 APPLICATION USAGE

57168 The *rewinddir()* function should be used in conjunction with *opendir()*, *readdir()*, and *closedir()* to
 57169 examine the contents of the directory. This method is recommended for portability.

57170 RATIONALE

57171 None.

57172 FUTURE DIRECTIONS

57173 None.

57174 SEE ALSO

57175 *closedir()*, *fdopendir()*, *readdir()*

57176 XBD *<dirent.h>*, *<sys/types.h>*

57177 CHANGE HISTORY

57178 First released in Issue 2.

57179 Issue 6

57180 In the SYNOPSIS, the optional include of the *<sys/types.h>* header is removed.

57181 The following new requirements on POSIX implementations derive from alignment with the
 57182 Single UNIX Specification:

- 57183 • The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was
 57184 required for conforming implementations of previous POSIX specifications, it was not
 57185 required for UNIX applications.

rint()*System Interfaces*57186 **NAME**

57187 rint, rintf, rintl — round-to-nearest integral value

57188 **SYNOPSIS**

```
57189 #include <math.h>
57190 double rint(double x);
57191 float rintf(float x);
57192 long double rintl(long double x);
```

57193 **DESCRIPTION**

57194 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 57195 conflict between the requirements described here and the ISO C standard is unintentional. This
 57196 volume of POSIX.1-2008 defers to the ISO C standard.

57197 These functions shall return the integral value (represented as a **double**) nearest x in the
 57198 direction of the current rounding mode. The current rounding mode is implementation-defined.

57199 If the current rounding mode rounds toward negative infinity, then *rint()* shall be equivalent to
 57200 *floor()*. If the current rounding mode rounds toward positive infinity, then *rint()* shall be
 57201 equivalent to *ceil()*.

57202 These functions differ from the *nearbyint()*, *nearbyintf()*, and *nearbyintl()* functions only in that
 57203 they may raise the inexact floating-point exception if the result differs in value from the
 57204 argument.

57205 An application wishing to check for error situations should set *errno* to zero and call
 57206 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 57207 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 57208 zero, an error has occurred.

57209 **RETURN VALUE**

57210 Upon successful completion, these functions shall return the integer (represented as a double
 57211 precision number) nearest x in the direction of the current rounding mode.

57212 MX If x is NaN, a NaN shall be returned.

57213 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

57214 XSI If the correct value would cause overflow, a range error shall occur and *rint()*, *rintf()*, and *rintl()*
 57215 shall return the value of the macro $\pm \text{HUGE\_VAL}$, $\pm \text{HUGE\_VALF}$, and $\pm \text{HUGE\_VALL}$ (with the
 57216 same sign as x), respectively.

57217 **ERRORS**

57218 These functions shall fail if:

57219 XSI **Range Error** The result would cause an overflow.

57220 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 57221 then *errno* shall be set to [ERANGE]. If the integer expression
 57222 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 57223 floating-point exception shall be raised.

57224 **EXAMPLES**

57225 None.

57226 **APPLICATION USAGE**

57227 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 57228 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

57229 **RATIONALE**

57230 None.

57231 **FUTURE DIRECTIONS**

57232 None.

57233 **SEE ALSO**57234 *abs()*, *ceil()*, *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*, *nearbyint()*

57235 XBD Section 4.19 (on page 116), <math.h>

57236 **CHANGE HISTORY**

57237 First released in Issue 4, Version 2.

57238 **Issue 5**

57239 Moved from X/OPEN UNIX extension to BASE.

57240 **Issue 6**

57241 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 57242 • The *rintf()* and *rintl()* functions are added.
- 57243 • The *rint()* function is no longer marked as an extension.
- 57244 • The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 57245 revised to align with the ISO/IEC 9899:1999 standard.
- 57246 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard
 57247 are marked.

rmdir()*System Interfaces*57248 **NAME**57249 **rmdir** — remove a directory57250 **SYNOPSIS**57251 `#include <unistd.h>`57252 `int rmdir(const char *path);`57253 **DESCRIPTION**

57254 The *rmdir()* function shall remove a directory whose name is given by *path*. The directory shall
 57255 be removed only if it is an empty directory.

57256 If the directory is the root directory or the current working directory of any process, it is
 57257 unspecified whether the function succeeds, or whether it shall fail and set *errno* to [EBUSY].

57258 If *path* names a symbolic link, then *rmdir()* shall fail and set *errno* to [ENOTDIR].

57259 If the *path* argument refers to a path whose final component is either dot or dot-dot, *rmdir()* shall
 57260 fail.

57261 If the directory's link count becomes 0 and no process has the directory open, the space occupied
 57262 by the directory shall be freed and the directory shall no longer be accessible. If one or more
 57263 processes have the directory open when the last link is removed, the dot and dot-dot entries, if
 57264 present, shall be removed before *rmdir()* returns and no new entries may be created in the
 57265 directory, but the directory shall not be removed until all references to the directory are closed.

57266 If the directory is not an empty directory, *rmdir()* shall fail and set *errno* to [EEXIST] or
 57267 [ENOTEMPTY].

57268 Upon successful completion, *rmdir()* shall mark for update the last data modification and last
 57269 file status change timestamps of the parent directory.

57270 **RETURN VALUE**

57271 Upon successful completion, the function *rmdir()* shall return 0. Otherwise, -1 shall be returned,
 57272 and *errno* set to indicate the error. If -1 is returned, the named directory shall not be changed.

57273 **ERRORS**

57274 The *rmdir()* function shall fail if:

57275 [EACCES] Search permission is denied on a component of the path prefix, or write
 57276 permission is denied on the parent directory of the directory to be removed.

57277 [EBUSY] The directory to be removed is currently in use by the system or some process
 57278 and the implementation considers this to be an error.

57279 [EEXIST] or [ENOTEMPTY]

57280 The *path* argument names a directory that is not an empty directory, or there
 57281 are hard links to the directory other than dot or a single entry in dot-dot.

57282 [EINVAL] The *path* argument contains a last component that is dot.

57283 [EIO] A physical I/O error has occurred.

57284 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 57285 argument.

57286 [ENAMETOOLONG]

57287 The length of a component of a pathname is longer than {NAME_MAX}.

57288 [ENOENT] A component of *path* does not name an existing file, or the *path* argument
 57289 names a nonexistent directory or points to an empty string.

57290	[ENOTDIR]	A component of <i>path</i> is not a directory.
57291	XSI	[EPERM] or [EACCES]
57292		The S_ISVTX flag is set on the directory containing the file referred to by the
57293		<i>path</i> argument and the process does not satisfy the criteria specified in XBD
57294		Section 4.2 (on page 107).
57295	[EROFS]	The directory entry to be removed resides on a read-only file system.
57296		The <i>rmdir()</i> function may fail if:
57297	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
57298		resolution of the <i>path</i> argument.
57299	[ENAMETOOLONG]	
57300		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
57301		symbolic link produced an intermediate result with a length that exceeds
57302		{PATH_MAX}.

EXAMPLES**Removing a Directory**

The following example shows how to remove a directory named */home/cnd/mod1*.

```
#include <unistd.h>

int status;
...
status = rmdir("/home/cnd/mod1");
```

APPLICATION USAGE

None.

RATIONALE

The *rmdir()* and *rename()* functions originated in 4.2 BSD, and they used [ENOTEMPTY] for the condition when the directory to be removed does not exist or *new* already exists. When the 1984 /usr/group standard was published, it contained [EEXIST] instead. When these functions were adopted into System V, the 1984 /usr/group standard was used as a reference. Therefore, several existing applications and implementations support/use both forms, and no agreement could be reached on either value. All implementations are required to supply both [EEXIST] and [ENOTEMPTY] in *<errno.h>* with distinct values, so that applications can use both values in C-language *case* statements.

The meaning of deleting *pathname/dot* is unclear, because the name of the file (directory) in the parent directory to be removed is not clear, particularly in the presence of multiple links to a directory.

The POSIX.1-1990 standard was silent with regard to the behavior of *rmdir()* when there are multiple hard links to the directory being removed. The requirement to set *errno* to [EEXIST] or [ENOTEMPTY] clarifies the behavior in this case.

If the current working directory of the process is being removed, that should be an allowed error.

Virtually all existing implementations detect [ENOTEMPTY] or the case of dot-dot. The text in Section 2.3 (on page 477) about returning any one of the possible errors permits that behavior to continue. The [ELOOP] error may be returned if more than {SYMLOOP_MAX} symbolic links are encountered during resolution of the *path* argument.

rmdir()*System Interfaces*57333 **FUTURE DIRECTIONS**

57334 None.

57335 **SEE ALSO**57336 [Section 2.3](#) (on page 477), [mkdir\(\)](#), [remove\(\)](#), [rename\(\)](#), [unlink\(\)](#)57337 XBD [Section 4.2](#) (on page 107), [<unistd.h>](#)57338 **CHANGE HISTORY**

57339 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

57340 **Issue 6**57341 The following new requirements on POSIX implementations derive from alignment with the
57342 Single UNIX Specification:

- 57343 • The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.
- 57344 • The [EIO] mandatory error condition is added.
- 57345 • The [ELOOP] mandatory error condition is added.
- 57346 • A second [ENAMETOOLONG] is added as an optional error condition.

57347 The following changes were made to align with the IEEE P1003.1a draft standard:

- 57348 • The [ELOOP] optional error condition is added.

57349 **Issue 7**

57350 Austin Group Interpretation 1003.1-2001 #143 is applied.

57351 Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for
57352 operations when the S_ISVTX bit is set.

57353 Changes are made related to support for finegrained timestamps.

NAME

round, roundf, roundl — round to the nearest integer value in a floating-point format

SYNOPSIS

```
#include <math.h>

double round(double x);
float roundf(float x);
long double roundl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value in floating-point format, rounding halfway cases away from zero, regardless of the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ± 0 or $\pm \text{Inf}$, *x* shall be returned.

XSI If the correct value would cause overflow, a range error shall occur and *round()*, *roundf()*, and *roundl()* shall return the value of the macro $\pm \text{HUGE\_VAL}$, $\pm \text{HUGE\_VALF}$, and $\pm \text{HUGE\_VALL}$ (with the same sign as *x*), respectively.

ERRORS

These functions may fail if:

XSI **Range Error** The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

round()*System Interfaces*57394 **SEE ALSO**57395 *feclearexcept()*, *fetestexcept()*57396 XBD [Section 4.19](#) (on page 116), [<math.h>](#)57397 **CHANGE HISTORY**

57398 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

57399 **NAME**57400 `scalbn`, `scalblnf`, `scalblnl`, `scalbn`, `scalbnf`, `scalbnl` — compute exponent using FLT_RADIX57401 **SYNOPSIS**

```
57402 #include <math.h>
57403 double scalbn(double x, long n);
57404 float scalblnf(float x, long n);
57405 long double scalblnl(long double x, long n);
57406 double scalbn(double x, int n);
57407 float scalbnf(float x, int n);
57408 long double scalbnl(long double x, int n);
```

57409 **DESCRIPTION**

57410 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 57411 conflict between the requirements described here and the ISO C standard is unintentional. This
 57412 volume of POSIX.1-2008 defers to the ISO C standard.

57413 These functions shall compute $x * \text{FLT\_RADIX}^n$ efficiently, not normally by computing
 57414 FLT_RADIXⁿ explicitly.

57415 An application wishing to check for error situations should set *errno* to zero and call
 57416 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 57417 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 57418 zero, an error has occurred.

57419 **RETURN VALUE**

57420 Upon successful completion, these functions shall return $x * \text{FLT\_RADIX}^n$.

57421 If the result would cause overflow, a range error shall occur and these functions shall return
 57422 ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (according to the sign of *x*) as appropriate for
 57423 the return type of the function.

57424 If the correct value would cause underflow, and is not representable, a range error may occur,
 57425 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

57426 MX If *x* is NaN, a NaN shall be returned.

57427 If *x* is ±0 or ±Inf, *x* shall be returned.

57428 If *n* is 0, *x* shall be returned.

57429 If the correct value would cause underflow, and is representable, a range error may occur and
 57430 the correct value shall be returned.

57431 **ERRORS**

57432 These functions shall fail if:

57433 Range Error The result overflows.

57434 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 57435 then *errno* shall be set to [ERANGE]. If the integer expression
 57436 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 57437 floating-point exception shall be raised.

57438 These functions may fail if:

57439 Range Error The result underflows.

57440 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 57441 then *errno* shall be set to [ERANGE]. If the integer expression

scalbln()*System Interfaces*

57442 *(math_errhandling & MATH_ERREXCEPT)* is non-zero, then the underflow
 57443 floating-point exception shall be raised.

57444 **EXAMPLES**

57445 None.

57446 **APPLICATION USAGE**

57447 On error, the expressions *(math_errhandling & MATH_ERRNO)* and *(math_errhandling &*
 57448 *MATH_ERREXCEPT)* are independent of each other, but at least one of them must be non-zero.

57449 **RATIONALE**

57450 These functions are named so as to avoid conflicting with the historical definition of the *scalb()*
 57451 function from the Single UNIX Specification. The difference is that the *scalb()* function has a
 57452 second argument of **double** instead of **int**. The *scalb()* function is not part of the ISO C standard.
 57453 The three functions whose second type is **long** are provided because the factor required to scale
 57454 from the smallest positive floating-point value to the largest finite one, on many
 57455 implementations, is too large to represent in the minimum-width **int** format.

57456 **FUTURE DIRECTIONS**

57457 None.

57458 **SEE ALSO**

57459 *feclearexcept()*, *fetestexcept()*

57460 XBD Section 4.19 (on page 116), **<math.h>**

57461 **CHANGE HISTORY**

57462 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

*System Interfaces***scandir()**57463 **NAME**

57464 scandir — scan a directory

57465 **SYNOPSIS**

```
57466       #include <dirent.h>

57467       int scandir(const char *dir, struct dirent ***namelist,
57468                   int (*sel)(const struct dirent *),
57469                   int (*compar)(const struct dirent **, const struct dirent **));
```

57470 **DESCRIPTION**57471 Refer to *alphasort()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

scanf()*System Interfaces*57472 **NAME**

57473 scanf — convert formatted input

57474 **SYNOPSIS**

57475 #include <stdio.h>

57476 int scanf(const char *restrict *format*, ...);57477 **DESCRIPTION**57478 Refer to *fscanf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

57479 NAME

57480 sched_get_priority_max, sched_get_priority_min — get priority limits (REALTIME)

57481 SYNOPSIS

```
57482 PS|TPS #include <sched.h>
57483 int sched_get_priority_max(int policy);
57484 int sched_get_priority_min(int policy);
```

57485 DESCRIPTION

57486 The *sched_get_priority_max()* and *sched_get_priority_min()* functions shall return the appropriate
 57487 maximum or minimum, respectively, for the scheduling policy specified by *policy*.

57488 The value of *policy* shall be one of the scheduling policy values defined in <sched.h>.

57489 RETURN VALUE

57490 If successful, the *sched_get_priority_max()* and *sched_get_priority_min()* functions shall return the
 57491 appropriate maximum or minimum values, respectively. If unsuccessful, they shall return a
 57492 value of -1 and set *errno* to indicate the error.

57493 ERRORS

57494 The *sched_get_priority_max()* and *sched_get_priority_min()* functions shall fail if:

57495 [EINVAL] The value of the *policy* parameter does not represent a defined scheduling
 57496 policy.

57497 EXAMPLES

57498 None.

57499 APPLICATION USAGE

57500 None.

57501 RATIONALE

57502 None.

57503 FUTURE DIRECTIONS

57504 None.

57505 SEE ALSO

57506 *sched_getparam()*, *sched_setparam()*, *sched_getscheduler()*, *sched_rr_get_interval()*,
 57507 *sched_setscheduler()*

57508 XBD <sched.h>

57509 CHANGE HISTORY

57510 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57511 Issue 6

57512 These functions are marked as part of the Process Scheduling option.

57513 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 57514 implementation does not support the Process Scheduling option.

57515 The [ESRCH] error condition has been removed since these functions do not take a *pid*
 57516 argument.

57517 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/52 is applied, changing the PS margin
 57518 code in the SYNOPSIS to PS|TPS.

sched_getparam()

System Interfaces

57519 **NAME**57520 sched_getparam — get scheduling parameters (**REALTIME**)57521 **SYNOPSIS**

```
57522 PS      #include <sched.h>
57523      int sched_getparam(pid_t pid, struct sched_param *param);
```

57524 **DESCRIPTION**

57525 The *sched_getparam()* function shall return the scheduling parameters of a process specified by
 57526 *pid* in the **sched_param** structure pointed to by *param*.

57527 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
 57528 parameters for the process whose process ID is equal to *pid* shall be returned.

57529 If *pid* is zero, the scheduling parameters for the calling process shall be returned. The behavior of
 57530 the *sched_getparam()* function is unspecified if the value of *pid* is negative.

57531 **RETURN VALUE**

57532 Upon successful completion, the *sched_getparam()* function shall return zero. If the call to
 57533 *sched_getparam()* is unsuccessful, the function shall return a value of -1 and set *errno* to indicate
 57534 the error.

57535 **ERRORS**

57536 The *sched_getparam()* function shall fail if:

57537 [EPERM] The requesting process does not have permission to obtain the scheduling
 57538 parameters of the specified process.

57539 [ESRCH] No process can be found corresponding to that specified by *pid*.

57540 **EXAMPLES**

57541 None.

57542 **APPLICATION USAGE**

57543 None.

57544 **RATIONALE**

57545 None.

57546 **FUTURE DIRECTIONS**

57547 None.

57548 **SEE ALSO**

57549 *sched_getscheduler()*, *sched_setparam()*, *sched_setscheduler()*

57550 XBD *<sched.h>*

57551 **CHANGE HISTORY**

57552 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57553 **Issue 6**

57554 The *sched_getparam()* function is marked as part of the Process Scheduling option.

57555 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 57556 implementation does not support the Process Scheduling option.

NAME

`sched_getscheduler` — get scheduling policy (**REALTIME**)

SYNOPSIS

```
#include <sched.h>

int sched_getscheduler(pid_t pid);
```

DESCRIPTION

The `sched_getscheduler()` function shall return the scheduling policy of the process specified by `pid`. If the value of `pid` is negative, the behavior of the `sched_getscheduler()` function is unspecified.

The values that can be returned by `sched_getscheduler()` are defined in the `<sched.h>` header.

If a process specified by `pid` exists, and if the calling process has permission, the scheduling policy shall be returned for the process whose process ID is equal to `pid`.

If `pid` is zero, the scheduling policy shall be returned for the calling process.

RETURN VALUE

Upon successful completion, the `sched_getscheduler()` function shall return the scheduling policy of the specified process. If unsuccessful, the function shall return `-1` and set `errno` to indicate the error.

ERRORS

The `sched_getscheduler()` function shall fail if:

[EPERM]	The requesting process does not have permission to determine the scheduling policy of the specified process.
[ESRCH]	No process can be found corresponding to that specified by <code>pid</code> .

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[`sched\_getparam\(\)`](#), [`sched\_setparam\(\)`](#), [`sched\_setscheduler\(\)`](#)

XBD `<sched.h>`

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The `sched_getscheduler()` function is marked as part of the Process Scheduling option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.

sched_rr_get_interval()

System Interfaces

57596 **NAME**57597 sched_rr_get_interval — get execution time limits (**REALTIME**)57598 **SYNOPSIS**

```
57599 PS|TPS #include <sched.h>
57600 int sched_rr_get_interval(pid_t pid, struct timespec *interval);
```

57601 **DESCRIPTION**

57602 The *sched_rr_get_interval()* function shall update the **timespec** structure referenced by the
 57603 *interval* argument to contain the current execution time limit (that is, time quantum) for the
 57604 process specified by *pid*. If *pid* is zero, the current execution time limit for the calling process
 57605 shall be returned.

57606 **RETURN VALUE**

57607 If successful, the *sched_rr_get_interval()* function shall return zero. Otherwise, it shall return a
 57608 value of -1 and set *errno* to indicate the error.

57609 **ERRORS**

57610 The *sched_rr_get_interval()* function shall fail if:

57611 [ESRCH] No process can be found corresponding to that specified by *pid*.

57612 **EXAMPLES**

57613 None.

57614 **APPLICATION USAGE**

57615 None.

57616 **RATIONALE**

57617 None.

57618 **FUTURE DIRECTIONS**

57619 None.

57620 **SEE ALSO**

57621 *sched_getparam()*, *sched_get_priority_max()*, *sched_getscheduler()*, *sched_setparam()*,
 57622 *sched_setscheduler()*

57623 XBD **<sched.h>**

57624 **CHANGE HISTORY**

57625 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57626 **Issue 6**

57627 The *sched_rr_get_interval()* function is marked as part of the Process Scheduling option.

57628 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 57629 implementation does not support the Process Scheduling option.

57630 IEEE Std 1003.1-2001/Cor 1-2002, XSH/TC1/D6/53 is applied, changing the PS margin code in
 57631 the SYNOPSIS to PS|TPS.

NAME

sched_setparam — set scheduling parameters (**REALTIME**)

SYNOPSIS

```
PS    #include <sched.h>

57636 int sched_setparam(pid_t pid, const struct sched_param *param);
```

DESCRIPTION

The *sched_setparam()* function shall set the scheduling parameters of the process specified by *pid* to the values specified by the **sched_param** structure pointed to by *param*. The value of the *sched_priority* member in the **sched_param** structure shall be any integer within the inclusive priority range for the current scheduling policy of the process specified by *pid*. Higher numerical values for the priority represent higher priorities. If the value of *pid* is negative, the behavior of the *sched_setparam()* function is unspecified.

If a process specified by *pid* exists, and if the calling process has permission, the scheduling parameters shall be set for the process whose process ID is equal to *pid*.

If *pid* is zero, the scheduling parameters shall be set for the calling process.

The conditions under which one process has permission to change the scheduling parameters of another process are implementation-defined.

Implementations may require the requesting process to have appropriate privileges to set its own scheduling parameters or those of another process.

See [Scheduling Policies](#) (on page 501) for a description on how this function affects the scheduling of the threads within the target process.

SS If the current scheduling policy for the target process is not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC, the result is implementation-defined; this case includes the SCHED_OTHER policy.

SS The specified *sched_ss_repl_period* shall be greater than or equal to the specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

The value of *sched_ss_max_repl* shall be within the inclusive range [1, {SS_REPL_MAX}] for the function to succeed; if not, the function shall fail. It is unspecified whether the *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are rounded to align with the resolution of the clock being used.

This function is not atomic with respect to other threads in the process. Threads may continue to execute while this function call is in the process of changing the scheduling policy for the underlying kernel-scheduled entities used by the process contention scope threads.

RETURN VALUE

If successful, the *sched_setparam()* function shall return zero.

If the call to *sched_setparam()* is unsuccessful, the priority shall remain unchanged, and the function shall return a value of -1 and set *errno* to indicate the error.

ERRORS

The *sched_setparam()* function shall fail if:

[EINVAL] One or more of the requested scheduling parameters is outside the range defined for the scheduling policy of the specified *pid*.

sched_setparam()

System Interfaces

57673	[EPERM]	The requesting process does not have permission to set the scheduling parameters for the specified process, or does not have appropriate privileges to invoke <i>sched_setparam()</i> .
57674		
57675		
57676	[ESRCH]	No process can be found corresponding to that specified by <i>pid</i> .
57677	EXAMPLES	
57678	None.	
57679	APPLICATION USAGE	
57680	None.	
57681	RATIONALE	
57682	None.	
57683	FUTURE DIRECTIONS	
57684	None.	
57685	SEE ALSO	
57686		Scheduling Policies (on page 501), <i>sched_getparam()</i> , <i>sched_getscheduler()</i> , <i>sched_setscheduler()</i>
57687	XBD	<sched.h>
57688	CHANGE HISTORY	
57689		First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
57690	Issue 6	
57691		The <i>sched_setparam()</i> function is marked as part of the Process Scheduling option.
57692		The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.
57693		
57694		The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:
57695		
57696		• In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is added.
57697		
57698		• Sections describing two-level scheduling and atomicity of the function are added.
57699		The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.
57700		IEEE PASC Interpretation 1003.1 #100 is applied.
57701	Issue 7	
57702		Austin Group Interpretation 1003.1-2001 #061 is applied, updating the DESCRIPTION.
57703		Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the <i>sched_ss_repl_period</i> and <i>sched_ss_init_budget</i> values.
57704		

System Interfaces

sched_setscheduler()57705 **NAME**57706 sched_setscheduler — set scheduling policy and parameters (**REALTIME**)57707 **SYNOPSIS**

```
57708 PS      #include <sched.h>
57709
57709      int sched_setscheduler(pid_t pid, int policy,
57710                          const struct sched_param *param);
```

57711 **DESCRIPTION**

57712 The *sched_setscheduler()* function shall set the scheduling policy and scheduling parameters of
 57713 the process specified by *pid* to *policy* and the parameters specified in the **sched_param** structure
 57714 pointed to by *param*, respectively. The value of the *sched_priority* member in the **sched_param**
 57715 structure shall be any integer within the inclusive priority range for the scheduling policy
 57716 specified by *policy*. If the value of *pid* is negative, the behavior of the *sched_setscheduler()*
 57717 function is unspecified.

57718 The possible values for the *policy* parameter are defined in the **<sched.h>** header.

57719 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
 57720 policy and scheduling parameters shall be set for the process whose process ID is equal to *pid*.

57721 If *pid* is zero, the scheduling policy and scheduling parameters shall be set for the calling
 57722 process.

57723 The conditions under which one process has appropriate privileges to change the scheduling
 57724 parameters of another process are implementation-defined.

57725 Implementations may require that the requesting process have permission to set its own
 57726 scheduling parameters or those of another process. Additionally, implementation-defined
 57727 restrictions may apply as to the appropriate privileges required to set the scheduling policy of
 57728 the process, or the scheduling policy of another process, to a particular value.

57729 The *sched_setscheduler()* function shall be considered successful if it succeeds in setting the
 57730 scheduling policy and scheduling parameters of the process specified by *pid* to the values
 57731 specified by *policy* and the structure pointed to by *param*, respectively.

57732 See **Scheduling Policies** (on page 501) for a description on how this function affects the
 57733 scheduling of the threads within the target process.

57734 SS If the current scheduling policy for the target process is not SCHED_FIFO, SCHED_RR, or
 57735 SCHED_SPORADIC, the result is implementation-defined; this case includes the
 57736 SCHED_OTHER policy.

57737 SS The specified *sched_ss_repl_period* shall be greater than or equal to the specified
 57738 *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

57739 The value of *sched_ss_max_repl* shall be within the inclusive range [1, {SS_REPL_MAX}] for the
 57740 function to succeed; if not, the function shall fail. It is unspecified whether the
 57741 *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are
 57742 rounded to align with the resolution of the clock being used.

57743 This function is not atomic with respect to other threads in the process. Threads may continue to
 57744 execute while this function call is in the process of changing the scheduling policy and
 57745 associated scheduling parameters for the underlying kernel-scheduled entities used by the
 57746 process contention scope threads.

sched_setscheduler()

System Interfaces

57747 RETURN VALUE

57748 Upon successful completion, the function shall return the former scheduling policy of the
 57749 specified process. If the *sched_setscheduler()* function fails to complete successfully, the policy
 57750 and scheduling parameters shall remain unchanged, and the function shall return a value of -1
 57751 and set *errno* to indicate the error.

57752 ERRORS

57753 The *sched_setscheduler()* function shall fail if:

57754	[EINVAL]	The value of the <i>policy</i> parameter is invalid, or one or more of the parameters contained in <i>param</i> is outside the valid range for the specified scheduling policy.
57755		
57756		
57757	[EPERM]	The requesting process does not have permission to set either or both of the scheduling parameters or the scheduling policy of the specified process.
57758		
57759	[ESRCH]	No process can be found corresponding to that specified by <i>pid</i> .

57760 EXAMPLES

57761 None.

57762 APPLICATION USAGE

57763 None.

57764 RATIONALE

57765 None.

57766 FUTURE DIRECTIONS

57767 None.

57768 SEE ALSO

57769 [Scheduling Policies](#) (on page 501), *sched_getparam()*, *sched_getscheduler()*, *sched_setparam()*

57770 XBD [<sched.h>](#)

57771 CHANGE HISTORY

57772 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57773 Issue 6

57774 The *sched_setscheduler()* function is marked as part of the Process Scheduling option.

57775 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 57776 implementation does not support the Process Scheduling option.

57777 The following new requirements on POSIX implementations derive from alignment with the
 57778 Single UNIX Specification:

- 57779 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is
 57780 added.
- 57781 • Sections describing two-level scheduling and atomicity of the function are added.

57782 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

57783 Issue 7

57784 Austin Group Interpretation 1003.1-2001 #061 is applied, updating the DESCRIPTION.

57785 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements
 57786 for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

57787 NAME

57788 sched_yield — yield the processor

57789 SYNOPSIS

57790 #include <sched.h>

57791 int sched_yield(void);

57792 DESCRIPTION

57793 The *sched_yield()* function shall force the running thread to relinquish the processor until it again
57794 becomes the head of its thread list. It takes no arguments.

57795 RETURN VALUE

57796 The *sched_yield()* function shall return 0 if it completes successfully; otherwise, it shall return a
57797 value of -1 and set *errno* to indicate the error.

57798 ERRORS

57799 No errors are defined.

57800 EXAMPLES

57801 None.

57802 APPLICATION USAGE

57803 The conceptual model for scheduling semantics in POSIX.1-2008 defines a set of thread lists. This
57804 set of thread lists is always present regardless of the scheduling options supported by the
57805 system. On a system where the Process Scheduling option is not supported, portable
57806 applications should not make any assumptions regarding whether threads from other processes
57807 will be on the same thread list.

57808 RATIONALE

57809 None.

57810 FUTURE DIRECTIONS

57811 None.

57812 SEE ALSO

57813 XBD <sched.h>

57814 CHANGE HISTORY

57815 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
57816 POSIX Threads Extension.

57817 Issue 6

57818 The *sched_yield()* function is now marked as part of the Process Scheduling and Threads options.

57819 Issue 7

57820 SD5-XSH-ERN-120 is applied, adding APPLICATION USAGE.

57821 The *sched_yield()* function is moved to the Base.

seed48()*System Interfaces*57822 **NAME**

57823 seed48 — seed a uniformly distributed pseudo-random non-negative long integer generator

57824 **SYNOPSIS**57825 XSI `#include <stdlib.h>`57826 `unsigned short *seed48(unsigned short seed16v[3]);`57827 **DESCRIPTION**57828 Refer to *drand48()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

57829 **NAME**

57830 seekdir — set the position of a directory stream

57831 **SYNOPSIS**

```
57832 XSI      #include <dirent.h>
57833         void seekdir(DIR *dirp, long loc);
```

57834 **DESCRIPTION**

57835 The *seekdir()* function shall set the position of the next *readdir()* operation on the directory
 57836 stream specified by *dirp* to the position specified by *loc*. The value of *loc* should have been
 57837 returned from an earlier call to *telldir()* using the same directory stream. The new position
 57838 reverts to the one associated with the directory stream when *telldir()* was performed.

57839 If the value of *loc* was not obtained from an earlier call to *telldir()*, or if a call to *rewinddir()*
 57840 occurred between the call to *telldir()* and the call to *seekdir()*, the results of subsequent calls to
 57841 *readdir()* are unspecified.

57842 **RETURN VALUE**57843 The *seekdir()* function shall not return a value.57844 **ERRORS**

57845 No errors are defined.

57846 **EXAMPLES**

57847 None.

57848 **APPLICATION USAGE**

57849 None.

57850 **RATIONALE**

57851 The original standard developers perceived that there were restrictions on the use of the
 57852 *seekdir()* and *telldir()* functions related to implementation details, and for that reason these
 57853 functions need not be supported on all POSIX-conforming systems. They are required on
 57854 implementations supporting the XSI option.

57855 One of the perceived problems of implementation is that returning to a given point in a directory
 57856 is quite difficult to describe formally, in spite of its intuitive appeal, when systems that use B-
 57857 trees, hashing functions, or other similar mechanisms to order their directories are considered.
 57858 The definition of *seekdir()* and *telldir()* does not specify whether, when using these interfaces, a
 57859 given directory entry will be seen at all, or more than once.

57860 On systems not supporting these functions, their capability can sometimes be accomplished by
 57861 saving a filename found by *readdir()* and later using *rewinddir()* and a loop on *readdir()* to
 57862 relocate the position from which the filename was saved.

57863 **FUTURE DIRECTIONS**

57864 None.

57865 **SEE ALSO**57866 *fdopendir()*, *readdir()*, *telldir()*57867 XBD *<dirent.h>*, *<sys/types.h>*57868 **CHANGE HISTORY**

57869 First released in Issue 2.

seekdir()*System Interfaces*57870 **Issue 6**57871 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.57872 **Issue 7**57873 SD5-XSH-ERN-200 is applied, updating the DESCRIPTION to note that the value of *loc* should
57874 have been returned from an earlier call to *telldir()* using the same directory stream.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***select()**57875 **NAME**

57876 select — synchronous I/O multiplexing

57877 **SYNOPSIS**

57878 #include <sys/select.h>

```
57879       int select(int nfds, fd_set *restrict readfds,  
57880                fd_set *restrict writefds, fd_set *restrict errorfds,  
57881                struct timeval *restrict timeout);
```

57882 **DESCRIPTION**57883 Refer to *pselect()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

sem_close()

System Interfaces

57884 NAME57885 `sem_close` — close a named semaphore**57886 SYNOPSIS**

57887 `#include <semaphore.h>`
 57888 `int sem_close(sem_t *sem);`

57889 DESCRIPTION

57890 The `sem_close()` function shall indicate that the calling process is finished using the named
 57891 semaphore indicated by `sem`. The effects of calling `sem_close()` for an unnamed semaphore (one
 57892 created by `sem_init()`) are undefined. The `sem_close()` function shall deallocate (that is, make
 57893 available for reuse by a subsequent `sem_open()` by this process) any system resources allocated
 57894 by the system for use by this process for this semaphore. The effect of subsequent use of the
 57895 semaphore indicated by `sem` by this process is undefined. If the semaphore has not been
 57896 removed with a successful call to `sem_unlink()`, then `sem_close()` has no effect on the state of the
 57897 semaphore. If the `sem_unlink()` function has been successfully invoked for `name` after the most
 57898 recent call to `sem_open()` with `O_CREAT` for this semaphore, then when all processes that have
 57899 opened the semaphore close it, the semaphore is no longer accessible.

57900 RETURN VALUE

57901 Upon successful completion, a value of zero shall be returned. Otherwise, a value of `-1` shall be
 57902 returned and `errno` set to indicate the error.

57903 ERRORS

57904 The `sem_close()` function may fail if:

57905 `[EINVAL]` The `sem` argument is not a valid semaphore descriptor.

57906 EXAMPLES

57907 None.

57908 APPLICATION USAGE

57909 The `sem_close()` function is part of the Semaphores option and need not be available on all
 57910 implementations.

57911 RATIONALE

57912 None.

57913 FUTURE DIRECTIONS

57914 None.

57915 SEE ALSO

57916 `semctl()`, `semget()`, `semop()`, `sem_init()`, `sem_open()`, `sem_unlink()`
 57917 XBD `<semaphore.h>`

57918 CHANGE HISTORY

57919 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57920 Issue 6

57921 The `sem_close()` function is marked as part of the Semaphores option.

57922 The `[ENOSYS]` error condition has been removed as stubs need not be provided if an
 57923 implementation does not support the Semaphores option.

57924 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/113 is applied, updating the ERRORS
 57925 section so that the `[EINVAL]` error becomes optional.

57926 **Issue 7**
57927

The *sem_close()* function is moved from the Semaphores option to the Base.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

sem_destroy()

System Interfaces

57928 **NAME**57929 `sem_destroy` — destroy an unnamed semaphore57930 **SYNOPSIS**

```
57931 #include <semaphore.h>
57932 int sem_destroy(sem_t *sem);
```

57933 **DESCRIPTION**

57934 The `sem_destroy()` function shall destroy the unnamed semaphore indicated by `sem`. Only a
 57935 semaphore that was created using `sem_init()` may be destroyed using `sem_destroy()`; the effect of
 57936 calling `sem_destroy()` with a named semaphore is undefined. The effect of subsequent use of the
 57937 semaphore `sem` is undefined until `sem` is reinitialized by another call to `sem_init()`.

57938 It is safe to destroy an initialized semaphore upon which no threads are currently blocked. The
 57939 effect of destroying a semaphore upon which other threads are currently blocked is undefined.

57940 **RETURN VALUE**

57941 Upon successful completion, a value of zero shall be returned. Otherwise, a value of `-1` shall be
 57942 returned and `errno` set to indicate the error.

57943 **ERRORS**

57944 The `sem_destroy()` function may fail if:

57945	[EINVAL]	The <code>sem</code> argument is not a valid semaphore.
57946	[EBUSY]	There are currently processes blocked on the semaphore.

57947 **EXAMPLES**

57948 None.

57949 **APPLICATION USAGE**

57950 The `sem_destroy()` function is part of the Semaphores option and need not be available on all
 57951 implementations.

57952 **RATIONALE**

57953 None.

57954 **FUTURE DIRECTIONS**

57955 None.

57956 **SEE ALSO**

57957 `semctl()`, `semget()`, `semop()`, `sem_init()`, `sem_open()`

57958 XBD `<semaphore.h>`

57959 **CHANGE HISTORY**

57960 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57961 **Issue 6**

57962 The `sem_destroy()` function is marked as part of the Semaphores option.

57963 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 57964 implementation does not support the Semaphores option.

57965 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/114 is applied, updating the ERRORS
 57966 section so that the [EINVAL] error becomes optional.

57967 **Issue 7**
57968

The *sem_destroy()* function is moved from the Semaphores option to the Base.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

sem_getvalue()

System Interfaces

57969 **NAME**57970 `sem_getvalue` — get the value of a semaphore57971 **SYNOPSIS**57972 `#include <semaphore.h>`57973 `int sem_getvalue(sem_t *restrict sem, int *restrict sval);`57974 **DESCRIPTION**

57975 The `sem_getvalue()` function shall update the location referenced by the `sval` argument to have
 57976 the value of the semaphore referenced by `sem` without affecting the state of the semaphore. The
 57977 updated value represents an actual semaphore value that occurred at some unspecified time
 57978 during the call, but it need not be the actual value of the semaphore when it is returned to the
 57979 calling process.

57980 If `sem` is locked, then the object to which `sval` points shall either be set to zero or to a negative
 57981 number whose absolute value represents the number of processes waiting for the semaphore at
 57982 some unspecified time during the call.

57983 **RETURN VALUE**

57984 Upon successful completion, the `sem_getvalue()` function shall return a value of zero. Otherwise,
 57985 it shall return a value of `-1` and set `errno` to indicate the error.

57986 **ERRORS**57987 The `sem_getvalue()` function may fail if:

57988 [EINVAL] The `sem` argument does not refer to a valid semaphore.

57989 **EXAMPLES**

57990 None.

57991 **APPLICATION USAGE**

57992 The `sem_getvalue()` function is part of the Semaphores option and need not be available on all
 57993 implementations.

57994 **RATIONALE**

57995 None.

57996 **FUTURE DIRECTIONS**

57997 None.

57998 **SEE ALSO**57999 `semctl()`, `semget()`, `semop()`, `sem_post()`, `sem_timedwait()`, `sem_trywait()`58000 XBD `<semaphore.h>`58001 **CHANGE HISTORY**

58002 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

58003 **Issue 6**58004 The `sem_getvalue()` function is marked as part of the Semaphores option.

58005 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 58006 implementation does not support the Semaphores option.

58007 The `sem_timedwait()` function is added to the SEE ALSO section for alignment with IEEE Std
 58008 1003.1d-1999.

58009 The **restrict** keyword is added to the `sem_getvalue()` prototype for alignment with the
 58010 ISO/IEC 9899:1999 standard.

58011 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/54 is applied.

*System Interfaces***sem_getvalue()**

- 58012 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/115 is applied, updating the ERRORS
58013 section so that the [EINVAL] error becomes optional.
- 58014 **Issue 7**
58015 The *sem_getvalue()* function is moved from the Semaphores option to the Base.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

sem_init()*System Interfaces*58016 **NAME**58017 `sem_init` — initialize an unnamed semaphore58018 **SYNOPSIS**

```
58019 #include <semaphore.h>
58020 int sem_init(sem_t *sem, int pshared, unsigned value);
```

58021 **DESCRIPTION**

58022 The `sem_init()` function shall initialize the unnamed semaphore referred to by *sem*. The value of
 58023 the initialized semaphore shall be *value*. Following a successful call to `sem_init()`, the semaphore
 58024 may be used in subsequent calls to `sem_wait()`, `sem_timedwait()`, `sem_trywait()`, `sem_post()`, and
 58025 `sem_destroy()`. This semaphore shall remain usable until the semaphore is destroyed.

58026 If the *pshared* argument has a non-zero value, then the semaphore is shared between processes;
 58027 in this case, any process that can access the semaphore *sem* can use *sem* for performing
 58028 `sem_wait()`, `sem_timedwait()`, `sem_trywait()`, `sem_post()`, and `sem_destroy()` operations.

58029 Only *sem* itself may be used for performing synchronization. The result of referring to copies of
 58030 *sem* in calls to `sem_wait()`, `sem_timedwait()`, `sem_trywait()`, `sem_post()`, and `sem_destroy()` is
 58031 undefined.

58032 If the *pshared* argument is zero, then the semaphore is shared between threads of the process; any
 58033 thread in this process can use *sem* for performing `sem_wait()`, `sem_timedwait()`, `sem_trywait()`,
 58034 `sem_post()`, and `sem_destroy()` operations. The use of the semaphore by threads other than those
 58035 created in the same process is undefined.

58036 Attempting to initialize an already initialized semaphore results in undefined behavior.

58037 **RETURN VALUE**

58038 Upon successful completion, the `sem_init()` function shall initialize the semaphore in *sem* and
 58039 return 0. Otherwise, it shall return `-1` and set *errno* to indicate the error.

58040 **ERRORS**

58041 The `sem_init()` function shall fail if:

58042	[EINVAL]	The <i>value</i> argument exceeds {SEM_VALUE_MAX}.
58043	[ENOSPC]	A resource required to initialize the semaphore has been exhausted, or the
58044		limit on semaphores ({SEM_NSEMS_MAX}) has been reached.
58045	[EPERM]	The process lacks appropriate privileges to initialize the semaphore.

58046 **EXAMPLES**

58047 None.

58048 **APPLICATION USAGE**

58049 The `sem_init()` function is part of the Semaphores option and need not be available on all
 58050 implementations.

58051 **RATIONALE**

58052 None.

58053 **FUTURE DIRECTIONS**

58054 None.

58055 **SEE ALSO**

58056 [*sem_destroy\(\)*](#), [*sem_post\(\)*](#), [*sem_timedwait\(\)*](#), [*sem_trywait\(\)*](#)

58057 XBD [*<semaphore.h>*](#)

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sem_init()* function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

The *sem_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/116 is applied, updating the DESCRIPTION to add the *sem_timedwait()* function for alignment with IEEE Std 1003.1d-1999.

Issue 7

SD5-XSH-ERN-176 is applied.

The *sem_init()* function is moved from the Semaphores option to the Base.

sem_open()*System Interfaces*58071 **NAME**58072 `sem_open` — initialize and open a named semaphore58073 **SYNOPSIS**58074 `#include <semaphore.h>`58075 `sem_t *sem_open(const char *name, int oflag, ...);`58076 **DESCRIPTION**

58077 The `sem_open()` function shall establish a connection between a named semaphore and a process.
 58078 Following a call to `sem_open()` with semaphore name *name*, the process may reference the
 58079 semaphore associated with *name* using the address returned from the call. This semaphore may
 58080 be used in subsequent calls to `sem_wait()`, `sem_timedwait()`, `sem_trywait()`, `sem_post()`, and
 58081 `sem_close()`. The semaphore remains usable by this process until the semaphore is closed by a
 58082 successful call to `sem_close()`, `_exit()`, or one of the *exec* functions.

58083 The *oflag* argument controls whether the semaphore is created or merely accessed by the call to
 58084 `sem_open()`. The following flag bits may be set in *oflag*:

58085 **O_CREAT** This flag is used to create a semaphore if it does not already exist. If **O_CREAT** is
 58086 set and the semaphore already exists, then **O_CREAT** has no effect, except as noted
 58087 under **O_EXCL**. Otherwise, `sem_open()` creates a named semaphore. The **O_CREAT**
 58088 flag requires a third and a fourth argument: *mode*, which is of type **mode_t**, and
 58089 *value*, which is of type **unsigned**. The semaphore is created with an initial value of
 58090 *value*. Valid initial values for semaphores are less than or equal to
 58091 {SEM_VALUE_MAX}.

58092 The user ID of the semaphore shall be set to the effective user ID of the process.
 58093 The group ID of the semaphore shall be set to the effective group ID of the process;
 58094 however, if the *name* argument is visible in the file system, the group ID may be set
 58095 to the group ID of the containing directory. The permission bits of the semaphore
 58096 are set to the value of the *mode* argument except those set in the file mode creation
 58097 mask of the process. When bits in *mode* other than the file permission bits are
 58098 specified, the effect is unspecified.

58099 After the semaphore named *name* has been created by `sem_open()` with the
 58100 **O_CREAT** flag, other processes can connect to the semaphore by calling
 58101 `sem_open()` with the same value of *name*.

58102 **O_EXCL** If **O_EXCL** and **O_CREAT** are set, `sem_open()` fails if the semaphore *name* exists.
 58103 The check for the existence of the semaphore and the creation of the semaphore if it
 58104 does not exist are atomic with respect to other processes executing `sem_open()` with
 58105 **O_EXCL** and **O_CREAT** set. If **O_EXCL** is set and **O_CREAT** is not set, the effect is
 58106 undefined.

58107 If flags other than **O_CREAT** and **O_EXCL** are specified in the *oflag* parameter, the
 58108 effect is unspecified.

58109 The *name* argument points to a string naming a semaphore object. It is unspecified whether the
 58110 name appears in the file system and is visible to functions that take pathnames as arguments.
 58111 The *name* argument conforms to the construction rules for a pathname, except that the
 58112 interpretation of <slash> characters other than the leading <slash> character in *name* is
 58113 implementation-defined, and that the length limits for the *name* argument are implementation-
 58114 defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If
 58115 *name* begins with the <slash> character, then processes calling `sem_open()` with the same value of
 58116 *name* shall refer to the same semaphore object, as long as that name has not been removed. If
 58117 *name* does not begin with the <slash> character, the effect is implementation-defined.

58118 If a process makes multiple successful calls to *sem_open()* with the same value for *name*, the same
 58119 semaphore address shall be returned for each such successful call, provided that there have been
 58120 no calls to *sem_unlink()* for this semaphore, and at least one previous successful *sem_open()* call
 58121 for this semaphore has not been matched with a *sem_close()* call.

58122 References to copies of the semaphore produce undefined results.

58123 RETURN VALUE

58124 Upon successful completion, the *sem_open()* function shall return the address of the semaphore.
 58125 Otherwise, it shall return a value of SEM_FAILED and set *errno* to indicate the error. The symbol
 58126 SEM_FAILED is defined in the **<semaphore.h>** header. No successful return from *sem_open()*
 58127 shall return the value SEM_FAILED.

58128 ERRORS

58129 If any of the following conditions occur, the *sem_open()* function shall return SEM_FAILED and
 58130 set *errno* to the corresponding value:

- 58131 [EACCES] The named semaphore exists and the permissions specified by *oflag* are
 58132 denied, or the named semaphore does not exist and permission to create the
 58133 named semaphore is denied.
- 58134 [EEXIST] O_CREAT and O_EXCL are set and the named semaphore already exists.
- 58135 [EINTR] The *sem_open()* operation was interrupted by a signal.
- 58136 [EINVAL] The *sem_open()* operation is not supported for the given name, or O_CREAT
 58137 was specified in *oflag* and *value* was greater than {SEM_VALUE_MAX}.
- 58138 [EMFILE] Too many semaphore descriptors or file descriptors are currently in use by this
 58139 process.
- 58140 [ENFILE] Too many semaphores are currently open in the system.
- 58141 [ENOENT] O_CREAT is not set and the named semaphore does not exist.
- 58142 [ENOMEM] There is insufficient memory for the creation of the new named semaphore.
- 58143 [ENOSPC] There is insufficient space on a storage device for the creation of the new
 58144 named semaphore.

58145 If any of the following conditions occur, the *sem_open()* function may return SEM_FAILED and
 58146 set *errno* to the corresponding value:

- 58147 [ENAMETOOLONG]
- 58148 The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems
 58149 XSI that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI
 58150 systems, or has a pathname component that is longer than
 58151 XSI {_POSIX_NAME_MAX} on systems that do not support the XSI option or
 58152 longer than {_XOPEN_NAME_MAX} on XSI systems.

sem_open()

System Interfaces

EXAMPLES

None.

APPLICATION USAGE

The *sem_open()* function is part of the Semaphores option and need not be available on all implementations.

RATIONALE

Early drafts required an error return value of `-1` with the type `sem_t *` for the *sem_open()* function, which is not guaranteed to be portable across implementations. The revised text provides the symbolic error code `SEM_FAILED` to eliminate the type conflict.

FUTURE DIRECTIONS

A future version might require the *sem_open()* and *sem_unlink()* functions to have semantics similar to normal file system operations.

SEE ALSO

semctl(), *semget()*, *semop()*, *sem_close()*, *sem_post()*, *sem_timedwait()*, *sem_trywait()*, *sem_unlink()*

XBD `<semaphore.h>`

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sem_open()* function is marked as part of the Semaphores option.

The `[ENOSYS]` error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

The *sem_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/117 is applied, updating the DESCRIPTION to add the *sem_timedwait()* function for alignment with IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/118 is applied, updating the DESCRIPTION to describe the conditions to return the same semaphore address on a call to *sem_open()*. The words “and at least one previous successful *sem_open()* call for this semaphore has not been matched with a *sem_close()* call” are added.

Issue 7

Austin Group Interpretation 1003.1-2001 #066 is applied, updating the `[ENOSPC]` error case and adding the `[ENOMEM]` error case.

Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and adding `[ENAMETOOLONG]` as a “may fail” error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the user ID and group ID of the semaphore.

The *sem_open()* function is moved from the Semaphores option to the Base.

NAME

sem_post — unlock a semaphore

SYNOPSIS

```
#include <semaphore.h>

int sem_post(sem_t *sem);
```

DESCRIPTION

The *sem_post()* function shall unlock the semaphore referenced by *sem* by performing a semaphore unlock operation on that semaphore.

If the semaphore value resulting from this operation is positive, then no threads were blocked waiting for the semaphore to become unlocked; the semaphore value is simply incremented.

If the value of the semaphore resulting from this operation is zero, then one of the threads blocked waiting for the semaphore shall be allowed to return successfully from its call to *sem_wait()*. If the Process Scheduling option is supported, the thread to be unblocked shall be chosen in a manner appropriate to the scheduling policies and parameters in effect for the blocked threads. In the case of the schedulers SCHED_FIFO and SCHED_RR, the highest priority waiting thread shall be unblocked, and if there is more than one highest priority thread blocked waiting for the semaphore, then the highest priority thread that has been waiting the longest shall be unblocked. If the Process Scheduling option is not defined, the choice of a thread to unblock is unspecified.

If the Process Sporadic Server option is supported, and the scheduling policy is SCHED_SPORADIC, the semantics are as per SCHED_FIFO above.

The *sem_post()* function shall be async-signal-safe and may be invoked from a signal-catching function.

RETURN VALUE

If successful, the *sem_post()* function shall return zero; otherwise, the function shall return -1 and set *errno* to indicate the error.

ERRORS

The *sem_post()* function may fail if:

[EINVAL] The *sem* argument does not refer to a valid semaphore.

EXAMPLES

See *sem_timedwait()*.

APPLICATION USAGE

The *sem_post()* function is part of the Semaphores option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

semctl(), *semget()*, *semop()*, *sem_timedwait()*, *sem_trywait()*

XBD Section 4.11 (on page 110), **<semaphore.h>**

sem_post()

System Interfaces

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sem_post()* function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

The *sem_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

SCHED_SPORADIC is added to the list of scheduling policies for which the thread that is to be unblocked is specified for alignment with IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/119 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

The *sem_post()* function is moved from the Semaphores option to the Base.

System Interfaces

sem_timedwait()**NAME**

sem_timedwait — lock a semaphore

SYNOPSIS

```
#include <semaphore.h>
#include <time.h>

int sem_timedwait(sem_t *restrict sem,
                  const struct timespec *restrict abstime);
```

DESCRIPTION

The *sem_timedwait()* function shall lock the semaphore referenced by *sem* as in the *sem_wait()* function. However, if the semaphore cannot be locked without waiting for another process or thread to unlock the semaphore by performing a *sem_post()* function, this wait shall be terminated when the specified timeout expires.

The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the clock on which it is based. The **timespec** data type is defined as a structure in the **<time.h>** header.

Under no circumstance shall the function fail with a timeout if the semaphore can be locked immediately. The validity of the *abstime* need not be checked if the semaphore can be locked immediately.

RETURN VALUE

The *sem_timedwait()* function shall return zero if the calling process successfully performed the semaphore lock operation on the semaphore designated by *sem*. If the call was unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a value of -1 and set *errno* to indicate the error.

ERRORS

The *sem_timedwait()* function shall fail if:

[EINVAL] The process or thread would have blocked, and the *abstime* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.

[ETIMEDOUT] The semaphore could not be locked before the specified timeout expired.

The *sem_timedwait()* function may fail if:

[EDEADLK] A deadlock condition was detected.

[EINTR] A signal interrupted this function.

[EINVAL] The *sem* argument does not refer to a valid semaphore.

sem_timedwait()**EXAMPLES**

The program shown below operates on an unnamed semaphore. The program expects two command-line arguments. The first argument specifies a seconds value that is used to set an alarm timer to generate a SIGALRM signal. This handler performs a *sem_post*(3) to increment the semaphore that is being waited on in *main*() using *sem_timedwait*(). The second command-line argument specifies the length of the timeout, in seconds, for *sem_timedwait*().

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <time.h>
#include <assert.h>
#include <errno.h>
#include <signal.h>

sem_t sem;

static void
handler(int sig)
{
    write(STDOUT_FILENO, "sem_post() from handler\n", 24);
    if (sem_post(&sem) == -1) {
        write(STDERR_FILENO, "sem_post() failed\n", 18);
        _exit(EXIT_FAILURE);
    }
}

int
main(int argc, char *argv[])
{
    struct sigaction sa;
    struct timespec ts;
    int s;

    if (argc != 3) {
        fprintf(stderr, "Usage: %s <alarm-secs> <wait-secs>\n",
            argv[0]);
        exit(EXIT_FAILURE);
    }

    if (sem_init(&sem, 0, 0) == -1) {
        perror("sem_init");
        exit(EXIT_FAILURE);
    }

    /* Establish SIGALRM handler; set alarm timer using argv[1] */

    sa.sa_handler = handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    if (sigaction(SIGALRM, &sa, NULL) == -1) {
        perror("sigaction");
        exit(EXIT_FAILURE);
    }
}
```

System Interfaces

sem_timedwait()

```

58331         alarm(atoi(argv[1]));
58332         /* Calculate relative interval as current time plus
58333            number of seconds given argv[2] */
58334         if (clock_gettime(CLOCK_REALTIME, &ts) == -1) {
58335             perror("clock_gettime");
58336             exit(EXIT_FAILURE);
58337         }
58338         ts.tv_sec += atoi(argv[2]);
58339         printf("main() about to call sem_timedwait()\n");
58340         while ((s = sem_timedwait(&sem, &ts)) == -1 && errno == EINTR)
58341             continue; /* Restart if interrupted by handler */
58342         /* Check what happened */
58343         if (s == -1) {
58344             if (errno == ETIMEDOUT)
58345                 printf("sem_timedwait() timed out\n");
58346             else
58347                 perror("sem_timedwait");
58348         } else
58349             printf("sem_timedwait() succeeded\n");
58350         exit((s == 0) ? EXIT_SUCCESS : EXIT_FAILURE);
58351     }

```

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.285 (on page 79).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[*sem_post\(\)*](#), [*sem_trywait\(\)*](#), [*semctl\(\)*](#), [*semget\(\)*](#), [*semop\(\)*](#), [*time\(\)*](#)

XBD Section 3.285 (on page 79), [**<semaphore.h>**](#), [**<time.h>**](#)

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/120 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

Issue 7

The *sem_timedwait()* function is moved from the Semaphores option to the Base.

Functionality relating to the Timers option is moved to the Base.

An example is added.

sem_trywait()*System Interfaces***58370 NAME**

58371 `sem_trywait`, `sem_wait` — lock a semaphore

58372 SYNOPSIS

```
58373 #include <semaphore.h>
58374 int sem_trywait(sem_t *sem);
58375 int sem_wait(sem_t *sem);
```

58376 DESCRIPTION

58377 The `sem_trywait()` function shall lock the semaphore referenced by `sem` only if the semaphore is
 58378 currently not locked; that is, if the semaphore value is currently positive. Otherwise, it shall not
 58379 lock the semaphore.

58380 The `sem_wait()` function shall lock the semaphore referenced by `sem` by performing a semaphore
 58381 lock operation on that semaphore. If the semaphore value is currently zero, then the calling
 58382 thread shall not return from the call to `sem_wait()` until it either locks the semaphore or the call is
 58383 interrupted by a signal.

58384 Upon successful return, the state of the semaphore shall be locked and shall remain locked until
 58385 the `sem_post()` function is executed and returns successfully.

58386 The `sem_wait()` function is interruptible by the delivery of a signal.

58387 RETURN VALUE

58388 The `sem_trywait()` and `sem_wait()` functions shall return zero if the calling process successfully
 58389 performed the semaphore lock operation on the semaphore designated by `sem`. If the call was
 58390 unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a
 58391 value of `-1` and set `errno` to indicate the error.

58392 ERRORS

58393 The `sem_trywait()` function shall fail if:

58394 [EAGAIN] The semaphore was already locked, so it cannot be immediately locked by the
 58395 `sem_trywait()` operation.

58396 The `sem_trywait()` and `sem_wait()` functions may fail if:

58397 [EDEADLK] A deadlock condition was detected.

58398 [EINTR] A signal interrupted this function.

58399 [EINVAL] The `sem` argument does not refer to a valid semaphore.

58400 EXAMPLES

58401 None.

58402 APPLICATION USAGE

58403 Applications using these functions may be subject to priority inversion, as discussed in XBD
 58404 [Section 3.285](#) (on page 79).

58405 The `sem_trywait()` and `sem_wait()` functions are part of the Semaphores option and need not be
 58406 provided on all implementations.

58407 RATIONALE

58408 None.

58409 FUTURE DIRECTIONS

58410 None.

SEE ALSO

semctl(), *semget()*, *semop()*, *sem_post()*, *sem_timedwait()*

XBD Section 3.285 (on page 79), Section 4.11 (on page 110), <semaphore.h>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sem_trywait()* and *sem_wait()* functions are marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

The *sem_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/121 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

Issue 7

SD5-XSH-ERN-54 is applied, removing the *sem_wait()* function from the “shall fail” error cases.

The *sem_trywait()* and *sem_wait()* functions are moved from the Semaphores option to the Base.

sem_unlink()*System Interfaces*58427 **NAME**58428 `sem_unlink` — remove a named semaphore58429 **SYNOPSIS**58430 `#include <semaphore.h>`58431 `int sem_unlink(const char *name);`58432 **DESCRIPTION**

58433 The `sem_unlink()` function shall remove the semaphore named by the string *name*. If the
 58434 semaphore named by *name* is currently referenced by other processes, then `sem_unlink()` shall
 58435 have no effect on the state of the semaphore. If one or more processes have the semaphore open
 58436 when `sem_unlink()` is called, destruction of the semaphore is postponed until all references to the
 58437 semaphore have been destroyed by calls to `sem_close()`, `_exit()`, or `exec`. Calls to `sem_open()` to
 58438 recreate or reconnect to the semaphore refer to a new semaphore after `sem_unlink()` is called. The
 58439 `sem_unlink()` call shall not block until all references have been destroyed; it shall return
 58440 immediately.

58441 **RETURN VALUE**

58442 Upon successful completion, the `sem_unlink()` function shall return a value of 0. Otherwise, the
 58443 semaphore shall not be changed and the function shall return a value of -1 and set *errno* to
 58444 indicate the error.

58445 **ERRORS**58446 The `sem_unlink()` function shall fail if:

58447 [EACCES] Permission is denied to unlink the named semaphore.

58448 [ENOENT] The named semaphore does not exist.

58449 The `sem_unlink()` function may fail if:

58450 [ENAMETOOLONG]

58451 The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems
 58452 XSI that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI
 58453 systems, or has a pathname component that is longer than
 58454 XSI {_POSIX_NAME_MAX} on systems that do not support the XSI option or
 58455 longer than {_XOPEN_NAME_MAX} on XSI systems. A call to `sem_unlink()`
 58456 with a *name* argument that contains the same semaphore name as was
 58457 previously used in a successful `sem_open()` call shall not give an
 58458 [ENAMETOOLONG] error.

58459 **EXAMPLES**

58460 None.

58461 **APPLICATION USAGE**

58462 The `sem_unlink()` function is part of the Semaphores option and need not be available on all
 58463 implementations.

58464 **RATIONALE**

58465 None.

58466 **FUTURE DIRECTIONS**

58467 A future version might require the `sem_open()` and `sem_unlink()` functions to have semantics
 58468 similar to normal file system operations.

58469 **SEE ALSO**58470 *semctl()*, *semget()*, *semop()*, *sem_close()*, *sem_open()*

58471 XBD <semaphore.h>

58472 **CHANGE HISTORY**

58473 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

58474 **Issue 6**58475 The *sem_unlink()* function is marked as part of the Semaphores option.58476 The [ENOSYS] error condition has been removed as stubs need not be provided if an
58477 implementation does not support the Semaphores option.58478 **Issue 7**58479 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a
58480 “shall fail” to a “may fail” error.

58481 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

58482 The *sem_unlink()* function is moved from the Semaphores option to the Base.

sem_wait()*System Interfaces*58483 **NAME**

58484 sem_wait — lock a semaphore

58485 **SYNOPSIS**

58486 #include <semaphore.h>

58487 int sem_wait(sem_t *sem);

58488 **DESCRIPTION**58489 Refer to *sem_trywait()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

semctl — XSI semaphore control operations

SYNOPSIS

```

#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, ...);

```

DESCRIPTION

The *semctl()* function operates on XSI semaphores (see XBD [Section 4.16](#), on page 113). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in [Section 2.8](#) (on page 497).

The *semctl()* function provides a variety of semaphore control operations as specified by *cmd*. The fourth argument is optional and depends upon the operation requested. If required, it is of type **union semun**, which the application shall explicitly declare:

```

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
} arg;

```

The following semaphore control operations as specified by *cmd* are executed with respect to the semaphore specified by *semid* and *semnum*. The level of permission required for each operation is shown with each command; see [Section 2.7](#) (on page 496). The symbolic names for the values of *cmd* are defined in the **<sys/sem.h>** header:

GETVAL Return the value of *semval*; see **<sys/sem.h>**. Requires read permission.

SETVAL Set the value of *semval* to *arg.val*, where *arg* is the value of the fourth argument to *semctl()*. When this command is successfully executed, the *semadj* value corresponding to the specified semaphore in all processes is cleared. Requires alter permission; see [Section 2.7](#) (on page 496).

GETPID Return the value of *sempid*. Requires read permission.

GETNCNT Return the value of *semmcnt*. Requires read permission.

GETZCNT Return the value of *semzcnt*. Requires read permission.

The following values of *cmd* operate on each *semval* in the set of semaphores:

GETALL Return the value of *semval* for each semaphore in the semaphore set and place into the array pointed to by *arg.array*, where *arg* is the fourth argument to *semctl()*. Requires read permission.

SETALL Set the value of *semval* for each semaphore in the semaphore set according to the array pointed to by *arg.array*, where *arg* is the fourth argument to *semctl()*. When this command is successfully executed, the *semadj* values corresponding to each specified semaphore in all processes are cleared. Requires alter permission.

The following values of *cmd* are also available:

IPC_STAT Place the current value of each member of the **semid_ds** data structure associated with *semid* into the structure pointed to by *arg.buf*, where *arg* is the fourth argument to *semctl()*. The contents of this structure are defined in **<sys/sem.h>**. Requires read permission.

semctl()*System Interfaces*

58533	IPC_SET	Set the value of the following members of the semid_ds data structure associated with <i>semid</i> to the corresponding value found in the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> :
58534		
58535		
58536		<i>sem_perm.uid</i>
58537		<i>sem_perm.gid</i>
58538		<i>sem_perm.mode</i>
58539		The mode bits specified in Section 2.7.1 (on page 496) are copied into the corresponding bits of the <i>sem_perm.mode</i> associated with <i>semid</i> . The stored values of any other bits are unspecified.
58540		
58541		
58542		This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the semid_ds data structure associated with <i>semid</i> .
58543		
58544		
58545		
58546	IPC_RMID	Remove the semaphore identifier specified by <i>semid</i> from the system and destroy the set of semaphores and semid_ds data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the semid_ds data structure associated with <i>semid</i> .
58547		
58548		
58549		
58550		
58551		
58552	RETURN VALUE	
58553	If successful, the value returned by <i>semctl()</i> depends on <i>cmd</i> as follows:	
58554	GETVAL	The value of <i>semval</i> .
58555	GETPID	The value of <i>sempid</i> .
58556	GETNCNT	The value of <i>semncnt</i> .
58557	GETZCNT	The value of <i>semzcnt</i> .
58558	All others	0.
58559	Otherwise, <i>semctl()</i> shall return -1 and set <i>errno</i> to indicate the error.	
58560	ERRORS	
58561	The <i>semctl()</i> function shall fail if:	
58562	[EACCES]	Operation permission is denied to the calling process; see Section 2.7 (on page 496).
58563		
58564	[EINVAL]	The value of <i>semid</i> is not a valid semaphore identifier, or the value of <i>semnum</i> is less than 0 or greater than or equal to <i>sem_nsems</i> , or the value of <i>cmd</i> is not a valid command.
58565		
58566		
58567	[EPERM]	The argument <i>cmd</i> is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the data structure associated with <i>semid</i> .
58568		
58569		
58570		
58571	[ERANGE]	The argument <i>cmd</i> is equal to SETVAL or SETALL and the value to which <i>semval</i> is to be set is greater than the system-imposed maximum.
58572		

EXAMPLES

None.

APPLICATION USAGE

The fourth parameter in the SYNOPSIS section is now specified as "... " in order to avoid a clash with the ISO C standard when referring to the union *semun* (as defined in Issue 3) and for backwards-compatibility.

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [semget\(\)](#), [semop\(\)](#), [sem_close\(\)](#), [sem_destroy\(\)](#), [sem_getvalue\(\)](#), [sem_init\(\)](#), [sem_open\(\)](#), [sem_post\(\)](#), [sem_trywait\(\)](#), [sem_unlink\(\)](#)

XBD [Section 4.16](#) (on page 113), [<sys/sem.h>](#)

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to the APPLICATION USAGE section.

semget()

System Interfaces

58596 **NAME**58597 **semget** — get set of XSI semaphores58598 **SYNOPSIS**

```
58599 XSI      #include <sys/sem.h>
58600      int semget(key_t key, int nsems, int semflg);
```

58601 **DESCRIPTION**

58602 The *semget()* function operates on XSI semaphores (see XBD [Section 4.16](#), on page 113). It is
 58603 unspecified whether this function interoperates with the realtime interprocess communication
 58604 facilities defined in [Section 2.8](#) (on page 497).

58605 The *semget()* function shall return the semaphore identifier associated with *key*.

58606 A semaphore identifier with its associated **semid_ds** data structure and its associated set of
 58607 *nsems* semaphores (see **<sys/sem.h>**) is created for *key* if one of the following is true:

- 58608 • The argument *key* is equal to `IPC_PRIVATE`.
- 58609 • The argument *key* does not already have a semaphore identifier associated with it and
 58610 (*semflg* & `IPC_CREAT`) is non-zero.

58611 Upon creation, the **semid_ds** data structure associated with the new semaphore identifier is
 58612 initialized as follows:

- 58613 • In the operation permissions structure *sem_perm.cuid*, *sem_perm.uid*, *sem_perm.cgid*, and
 58614 *sem_perm.gid* shall be set equal to the effective user ID and effective group ID, respectively,
 58615 of the calling process.
- 58616 • The low-order 9 bits of *sem_perm.mode* shall be set equal to the low-order 9 bits of *semflg*.
- 58617 • The variable *sem_nsems* shall be set equal to the value of *nsems*.
- 58618 • The variable *sem_otime* shall be set equal to 0 and *sem_ctime* shall be set equal to the current
 58619 time.
- 58620 • The data structure associated with each semaphore in the set need not be initialized. The
 58621 *semctl()* function with the command `SETVAL` or `SETALL` can be used to initialize each
 58622 semaphore.

58623 **RETURN VALUE**

58624 Upon successful completion, *semget()* shall return a non-negative integer, namely a semaphore
 58625 identifier; otherwise, it shall return -1 and set *errno* to indicate the error.

58626 **ERRORS**

58627 The *semget()* function shall fail if:

58628	[EACCES]	A semaphore identifier exists for <i>key</i> , but operation permission as specified by the low-order 9 bits of <i>semflg</i> would not be granted; see Section 2.7 (on page 496).
58629		
58630		
58631	[EEXIST]	A semaphore identifier exists for the argument <i>key</i> but ((<i>semflg</i> & <code>IPC_CREAT</code>) && (<i>semflg</i> & <code>IPC_EXCL</code>)) is non-zero.
58632		
58633	[EINVAL]	The value of <i>nsems</i> is either less than or equal to 0 or greater than the system-imposed limit, or a semaphore identifier exists for the argument <i>key</i> , but the number of semaphores in the set associated with it is less than <i>nsems</i> and <i>nsems</i> is not equal to 0.
58634		
58635		
58636		

58637	[ENOENT]	A semaphore identifier does not exist for the argument <i>key</i> and (<i>semflg</i> &IPC_CREAT) is equal to 0.
58638		
58639	[ENOSPC]	A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores system-wide would be exceeded.
58640		

58641 **EXAMPLES**58642 **Creating a Semaphore Identifier**

58643 The following example gets a unique semaphore key using the *ftok()* function, then gets a
 58644 semaphore ID associated with that key using the *semget()* function (the first call also tests to
 58645 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as
 58646 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the
 58647 program attempts to create one semaphore with read/write permission for all. It also uses the
 58648 IPC_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

58649 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in
 58650 the *sbuf* array. The number of processes that can execute concurrently without queuing is
 58651 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in
 58652 the program.

```

58653 #include <sys/types.h>
58654 #include <stdio.h>
58655 #include <sys/ipc.h>
58656 #include <sys/sem.h>
58657 #include <sys/stat.h>
58658 #include <errno.h>
58659 #include <unistd.h>
58660 #include <stdlib.h>
58661 #include <pwd.h>
58662 #include <fcntl.h>
58663 #include <limits.h>
58664 ...
58665 key_t semkey;
58666 int semid, pfd, fv;
58667 struct sembuf sbuf;
58668 char *lgn;
58669 char filename[PATH_MAX+1];
58670 struct stat outstat;
58671 struct passwd *pw;
58672 ...
58673 /* Get unique key for semaphore. */
58674 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
58675     perror("IPC error: ftok"); exit(1);
58676 }
58677
58678 /* Get semaphore ID associated with this key. */
58679 if ((semid = semget(semkey, 0, 0)) == -1) {
58679     /* Semaphore does not exist - Create. */
58680     if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
58681         S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
58682     {
58683         /* Initialize the semaphore. */

```

semget()*System Interfaces*

```

58684         sbuf.sem_num = 0;
58685         sbuf.sem_op = 2; /* This is the number of runs
58686                        without queuing. */
58687         sbuf.sem_flg = 0;
58688         if (semop(semid, &sbuf, 1) == -1) {
58689             perror("IPC error: semop"); exit(1);
58690         }
58691     }
58692     else if (errno == EEXIST) {
58693         if ((semid = semget(semkey, 0, 0)) == -1) {
58694             perror("IPC error 1: semget"); exit(1);
58695         }
58696     }
58697     else {
58698         perror("IPC error 2: semget"); exit(1);
58699     }
58700 }
58701 ...

```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [semctl\(\)](#), [semop\(\)](#), [sem_close\(\)](#), [sem_destroy\(\)](#), [sem_getvalue\(\)](#), [sem_init\(\)](#), [sem_open\(\)](#), [sem_post\(\)](#), [sem_trywait\(\)](#), [sem_unlink\(\)](#)

XBD [Section 4.16](#) (on page 113), [<sys/sem.h>](#)

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

Issue 6

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/122 is applied, updating the DESCRIPTION from “each semaphore in the set shall not be initialized” to “each semaphore in the set need not be initialized”.

NAME

semop — XSI semaphore operations

SYNOPSIS

```

#include <sys/sem.h>

int semop(int semid, struct sembuf *sops, size_t nsops);

```

DESCRIPTION

The *semop()* function operates on XSI semaphores (see XBD Section 4.16, on page 113). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 497).

The *semop()* function shall perform atomically a user-defined array of semaphore operations in array order on the set of semaphores associated with the semaphore identifier specified by the argument *semid*.

The argument *sops* is a pointer to a user-defined array of semaphore operation structures. The implementation shall not modify elements of this array unless the application uses implementation-defined extensions.

The argument *nsops* is the number of such structures in the array.

Each structure, **sembuf**, includes the following members:

Member Type	Member Name	Description
short	<i>sem_num</i>	Semaphore number.
short	<i>sem_op</i>	Semaphore operation.
short	<i>sem_flg</i>	Operation flags.

Each semaphore operation specified by *sem_op* is performed on the corresponding semaphore specified by *semid* and *sem_num*.

The variable *sem_op* specifies one of three semaphore operations:

1. If *sem_op* is a negative integer and the calling process has alter permission, one of the following shall occur:
 - If *semval* (see <sys/sem.h>) is greater than or equal to the absolute value of *sem_op*, the absolute value of *sem_op* is subtracted from *semval*. Also, if (*sem_flg* & SEM_UNDO) is non-zero, the absolute value of *sem_op* shall be added to the *semadj* value of the calling process for the specified semaphore.
 - If *semval* is less than the absolute value of *sem_op* and (*sem_flg* & IPC_NOWAIT) is non-zero, *semop()* shall return immediately.
 - If *semval* is less than the absolute value of *sem_op* and (*sem_flg* & IPC_NOWAIT) is 0, *semop()* shall increment the *semncnt* associated with the specified semaphore and suspend execution of the calling thread until one of the following conditions occurs:
 - The value of *semval* becomes greater than or equal to the absolute value of *sem_op*. When this occurs, the value of *semncnt* associated with the specified semaphore shall be decremented, the absolute value of *sem_op* shall be subtracted from *semval* and, if (*sem_flg* & SEM_UNDO) is non-zero, the absolute value of *sem_op* shall be added to the *semadj* value of the calling process for the specified semaphore.

semop()*System Interfaces*

- 58765 — The *semid* for which the calling thread is awaiting action is removed from the
 58766 system. When this occurs, *errno* shall be set equal to [EIDRM] and -1 shall be
 58767 returned.
- 58768 — The calling thread receives a signal that is to be caught. When this occurs, the
 58769 value of *semncnt* associated with the specified semaphore shall be
 58770 decremented, and the calling thread shall resume execution in the manner
 58771 prescribed in *sigaction()*.
- 58772 2. If *sem_op* is a positive integer and the calling process has alter permission, the value of
 58773 *sem_op* shall be added to *semval* and, if (*sem_flg* & SEM_UNDO) is non-zero, the value of
 58774 *sem_op* shall be subtracted from the *semadj* value of the calling process for the specified
 58775 semaphore.
- 58776 3. If *sem_op* is 0 and the calling process has read permission, one of the following shall occur:
- 58777 • If *semval* is 0, *semop()* shall return immediately.
- 58778 • If *semval* is non-zero and (*sem_flg* & IPC_NOWAIT) is non-zero, *semop()* shall return
 58779 immediately.
- 58780 • If *semval* is non-zero and (*sem_flg* & IPC_NOWAIT) is 0, *semop()* shall increment the
 58781 *semzcnt* associated with the specified semaphore and suspend execution of the
 58782 calling thread until one of the following occurs:
- 58783 — The value of *semval* becomes 0, at which time the value of *semzcnt* associated
 58784 with the specified semaphore shall be decremented.
- 58785 — The *semid* for which the calling thread is awaiting action is removed from the
 58786 system. When this occurs, *errno* shall be set equal to [EIDRM] and -1 shall be
 58787 returned.
- 58788 — The calling thread receives a signal that is to be caught. When this occurs, the
 58789 value of *semzcnt* associated with the specified semaphore shall be
 58790 decremented, and the calling thread shall resume execution in the manner
 58791 prescribed in *sigaction()*.
- 58792 Upon successful completion, the value of *sempid* for each semaphore specified in the array
 58793 pointed to by *sops* shall be set equal to the process ID of the calling process.

RETURN VALUE

58795 Upon successful completion, *semop()* shall return 0; otherwise, it shall return -1 and set *errno* to
 58796 indicate the error.

ERRORS

58797 The *semop()* function shall fail if:

- 58799 [E2BIG] The value of *nsops* is greater than the system-imposed maximum.
- 58800 [EACCES] Operation permission is denied to the calling process; see [Section 2.7](#) (on page
 58801 496).
- 58802 [EAGAIN] The operation would result in suspension of the calling process but (*sem_flg*
 58803 & IPC_NOWAIT) is non-zero.
- 58804 [EFBIG] The value of *sem_num* is less than 0 or greater than or equal to the number of
 58805 semaphores in the set associated with *semid*.

58806	[EIDRM]	The semaphore identifier <i>semid</i> is removed from the system.
58807	[EINTR]	The <i>semop()</i> function was interrupted by a signal.
58808	[EINVAL]	The value of <i>semid</i> is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a SEM_UNDO would exceed the system-imposed limit.
58809		
58810		
58811	[ENOSPC]	The limit on the number of individual processes requesting a SEM_UNDO would be exceeded.
58812		
58813	[ERANGE]	An operation would cause a <i>semval</i> to overflow the system-imposed limit, or an operation would cause a <i>semadj</i> value to overflow the system-imposed limit.
58814		
58815		

58816 EXAMPLES

58817 Setting Values in Semaphores

58818 The following example sets the values of the two semaphores associated with the *semid* identifier
 58819 to the values contained in the *sb* array.

```

58820 #include <sys/sem.h>
58821 ...
58822 int semid;
58823 struct sembuf sb[2];
58824 int nsops = 2;
58825 int result;

58826 /* Adjust value of semaphore in the semaphore array semid. */
58827 sb[0].sem_num = 0;
58828 sb[0].sem_op = -1;
58829 sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
58830 sb[1].sem_num = 1;
58831 sb[1].sem_op = 1;
58832 sb[1].sem_flg = 0;

58833 result = semop(semid, sb, nsops);

```

58834 Creating a Semaphore Identifier

58835 The following example gets a unique semaphore key using the *ftok()* function, then gets a
 58836 semaphore ID associated with that key using the *semget()* function (the first call also tests to
 58837 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as
 58838 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the
 58839 program attempts to create one semaphore with read/write permission for all. It also uses the
 58840 IPC_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

58841 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in
 58842 the *sbuf* array. The number of processes that can execute concurrently without queuing is
 58843 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in
 58844 the program.

58845 The final call to *semop()* acquires the semaphore and waits until it is free; the SEM_UNDO
 58846 option releases the semaphore when the process exits, waiting until there are less than two
 58847 processes running concurrently.

semop()*System Interfaces*

```

58848     #include <sys/types.h>
58849     #include <stdio.h>
58850     #include <sys/ipc.h>
58851     #include <sys/sem.h>
58852     #include <sys/stat.h>
58853     #include <errno.h>
58854     #include <unistd.h>
58855     #include <stdlib.h>
58856     #include <pwd.h>
58857     #include <fcntl.h>
58858     #include <limits.h>
58859     ...
58860     key_t semkey;
58861     int semid, pfd, fv;
58862     struct sembuf sbuf;
58863     char *lgn;
58864     char filename[PATH_MAX+1];
58865     struct stat outstat;
58866     struct passwd *pw;
58867     ...
58868     /* Get unique key for semaphore. */
58869     if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
58870         perror("IPC error: ftok"); exit(1);
58871     }
58872     /* Get semaphore ID associated with this key. */
58873     if ((semid = semget(semkey, 0, 0)) == -1) {
58874         /* Semaphore does not exist - Create. */
58875         if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
58876             S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
58877         {
58878             /* Initialize the semaphore. */
58879             sbuf.sem_num = 0;
58880             sbuf.sem_op = 2; /* This is the number of runs without queuing. */
58881             sbuf.sem_flg = 0;
58882             if (semop(semid, &sbuf, 1) == -1) {
58883                 perror("IPC error: semop"); exit(1);
58884             }
58885         }
58886         else if (errno == EEXIST) {
58887             if ((semid = semget(semkey, 0, 0)) == -1) {
58888                 perror("IPC error 1: semget"); exit(1);
58889             }
58890         }
58891         else {
58892             perror("IPC error 2: semget"); exit(1);
58893         }
58894     }
58895     ...
58896     sbuf.sem_num = 0;
58897     sbuf.sem_op = -1;
58898     sbuf.sem_flg = SEM_UNDO;

```

```

58899         if (semop(semid, &sbuf, 1) == -1) {
58900             perror("IPC Error: semop"); exit(1);
58901         }

```

58902 APPLICATION USAGE

58903 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
 58904 Application developers who need to use IPC should design their applications so that modules
 58905 using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to use the
 58906 alternative interfaces.

58907 RATIONALE

58908 None.

58909 FUTURE DIRECTIONS

58910 None.

58911 SEE ALSO

58912 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), *exec*, *exit()*, *fork()*, *semctl()*, *semget()*,
 58913 *sem_close()*, *sem_destroy()*, *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_trywait()*,
 58914 *sem_unlink()*

58915 XBD [Section 4.16](#) (on page 113), [<sys/ipc.h>](#), [<sys/sem.h>](#), [<sys/types.h>](#)

58916 CHANGE HISTORY

58917 First released in Issue 2. Derived from Issue 2 of the SVID.

58918 Issue 5

58919 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
 58920 DIRECTIONS to a new APPLICATION USAGE section.

58921 Issue 7

58922 SD5-XSH-ERN-171 is applied, updating the DESCRIPTION to clarify the order in which the
 58923 operations in *sops* will be performed when there are multiple operations.

send()*System Interfaces*58924 **NAME**58925 **send** — send a message on a socket58926 **SYNOPSIS**58927 `#include <sys/socket.h>`58928 `ssize_t send(int socket, const void *buffer, size_t length, int flags);`58929 **DESCRIPTION**

58930 The *send()* function shall initiate transmission of a message from the specified socket to its peer.
 58931 The *send()* function shall send a message only when the socket is connected. If the socket is a
 58932 connectionless-mode socket, the message shall be sent to the pre-specified peer address.

58933 The *send()* function takes the following arguments:

58934 *socket* Specifies the socket file descriptor.

58935 *buffer* Points to the buffer containing the message to send.

58936 *length* Specifies the length of the message in bytes.

58937 *flags* Specifies the type of message transmission. Values of this argument are
 58938 formed by logically OR'ing zero or more of the following flags:

58939 MSG_EOR Terminates a record (if supported by the protocol).

58940 MSG_OOB Sends out-of-band data on sockets that support out-of-
 58941 band communications. The significance and semantics
 58942 of out-of-band data are protocol-specific.

58943 MSG_NOSIGNAL Requests not to send the SIGPIPE signal if an attempt to
 58944 send is made on a stream-oriented socket that is no
 58945 longer connected. The [EPIPE] error shall still be
 58946 returned.

58947 The length of the message to be sent is specified by the *length* argument. If the message is too
 58948 long to pass through the underlying protocol, *send()* shall fail and no data shall be transmitted.

58949 Successful completion of a call to *send()* does not guarantee delivery of the message. A return
 58950 value of `-1` indicates only locally-detected errors.

58951 If space is not available at the sending socket to hold the message to be transmitted, and the
 58952 socket file descriptor does not have `O_NONBLOCK` set, *send()* shall block until space is
 58953 available. If space is not available at the sending socket to hold the message to be transmitted,
 58954 and the socket file descriptor does have `O_NONBLOCK` set, *send()* shall fail. The *select()* and
 58955 *poll()* functions can be used to determine when it is possible to send more data.

58956 The socket in use may require the process to have appropriate privileges to use the *send()*
 58957 function.

58958 **RETURN VALUE**

58959 Upon successful completion, *send()* shall return the number of bytes sent. Otherwise, `-1` shall be
 58960 returned and *errno* set to indicate the error.

58961 **ERRORS**

58962 The *send()* function shall fail if:

58963 [EAGAIN] or [EWOULDBLOCK]

58964 The socket's file descriptor is marked `O_NONBLOCK` and the requested
 58965 operation would block.

58966	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
58967	[ECONNRESET]	A connection was forcibly closed by a peer.
58968	[EDESTADDRREQ]	
58969		The socket is not connection-mode and no peer address is set.
58970	[EINTR]	A signal interrupted <i>send()</i> before any data was transmitted.
58971	[EMSGSIZE]	The message is too large to be sent all at once, as the socket requires.
58972	[ENOTCONN]	The socket is not connected.
58973	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
58974	[EOPNOTSUPP]	The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> .
58976	[EPIPE]	The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not set, the SIGPIPE signal is generated to the calling thread.
58980		The <i>send()</i> function may fail if:
58981	[EACCES]	The calling process does not have appropriate privileges.
58982	[EIO]	An I/O error occurred while reading from or writing to the file system.
58983	[ENETDOWN]	The local network interface used to reach the destination is down.
58984	[ENETUNREACH]	
58985		No route to the network is present.
58986	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
58987	EXAMPLES	
58988	None.	
58989	APPLICATION USAGE	
58990	The <i>send()</i> function is equivalent to <i>sendto()</i> with a null pointer <i>dest_len</i> argument, and to <i>write()</i> if no flags are used.	
58991		
58992	RATIONALE	
58993	None.	
58994	FUTURE DIRECTIONS	
58995	None.	
58996	SEE ALSO	
58997	<i>connect()</i> , <i>getsockopt()</i> , <i>poll()</i> , <i>pselect()</i> , <i>recv()</i> , <i>recvfrom()</i> , <i>recvmsg()</i> , <i>sendmsg()</i> , <i>sendto()</i> , <i>setsockopt()</i> , <i>shutdown()</i> , <i>socket()</i>	
58998		
58999	XBD <sys/socket.h>	
59000	CHANGE HISTORY	
59001	First released in Issue 6. Derived from the XNS, Issue 5.2 specification.	
59002	Issue 7	
59003	Austin Group Interpretation 1003.1-2001 #035 is applied, updating the DESCRIPTION to clarify the behavior when the socket is a connectionless-mode socket.	
59004		

send()*System Interfaces*

59005	The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.
59006	
59007	The [EPIPE] error is modified.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

sendmsg()**NAME**

sendmsg — send a message on a socket using a message structure

SYNOPSIS

```
#include <sys/socket.h>
```

```
ssize_t sendmsg(int socket, const struct msghdr *message, int flags);
```

DESCRIPTION

The *sendmsg()* function shall send a message through a connection-mode or connectionless-mode socket. If the socket is a connectionless-mode socket, the message shall be sent to the address specified by **msghdr** if no pre-specified peer address has been set. If a peer address has been pre-specified, either the message shall be sent to the address specified in **msghdr** (overriding the pre-specified peer address), or the function shall return **-1** and set *errno* to **[EISCONN]**. If the socket is connection-mode, the destination address in **msghdr** shall be ignored.

The *sendmsg()* function takes the following arguments:

<i>socket</i>	Specifies the socket file descriptor.
<i>message</i>	Points to a msghdr structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored.
<i>flags</i>	Specifies the type of message transmission. The application may specify 0 or the following flag:
MSG_EOR	Terminates a record (if supported by the protocol).
MSG_OOB	Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
MSG_NOSIGNAL	Requests not to send the SIGPIPE signal if an attempt to send is made on a stream-oriented socket that is no longer connected. The [EPIPE] error shall still be returned.

The *msg_iov* and *msg_iovlen* fields of *message* specify zero or more buffers containing the data to be sent. *msg_iov* points to an array of **iovec** structures; *msg_iovlen* shall be set to the dimension of this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated by *msg_iov* is sent in turn.

Successful completion of a call to *sendmsg()* does not guarantee delivery of the message. A return value of **-1** indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have **O_NONBLOCK** set, the *sendmsg()* function shall block until space is available. If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have **O_NONBLOCK** set, the *sendmsg()* function shall fail.

If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, *sendmsg()* shall fail if the **SO_BROADCAST** option is not set for the socket.

The socket in use may require the process to have appropriate privileges to use the *sendmsg()* function.

sendmsg()*System Interfaces***RETURN VALUE**

59052
59053 Upon successful completion, *sendmsg()* shall return the number of bytes sent. Otherwise, -1
59054 shall be returned and *errno* set to indicate the error.

ERRORS

59055
59056 The *sendmsg()* function shall fail if:

59057 [EAGAIN] or [EWOULDBLOCK]

59058 The socket's file descriptor is marked O_NONBLOCK and the requested
59059 operation would block.

59060 [EAFNOSUPPORT]

59061 Addresses in the specified address family cannot be used with this socket.

59062 [EBADF] The *socket* argument is not a valid file descriptor.

59063 [ECONNRESET] A connection was forcibly closed by a peer.

59064 [EINTR] A signal interrupted *sendmsg()* before any data was transmitted.

59065 [EINVAL] The sum of the *iov_len* values overflows an *ssize_t*.

59066 [EMSGSIZE] The message is too large to be sent all at once (as the socket requires), or the
59067 *msg_iovlen* member of the **msghdr** structure pointed to by *message* is less than
59068 or equal to 0 or is greater than {IOV_MAX}.

59069 [ENOTCONN] The socket is connection-mode but is not connected.

59070 [ENOTSOCK] The *socket* argument does not refer to a socket.

59071 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or
59072 more of the values set in *flags*.

59073 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is
59074 no longer connected. In the latter case, and if the socket is of type
59075 SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not
59076 set, the SIGPIPE signal is generated to the calling thread.

59077 If the address family of the socket is AF_UNIX, then *sendmsg()* shall fail if:

59078 [EIO] An I/O error occurred while reading from or writing to the file system.

59079 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
59080 in the socket address.

59081 [ENAMETOOLONG]

59082 The length of a component of a pathname is longer than {NAME_MAX}.

59083 [ENOENT]

59084 A component of the pathname does not name an existing file or the path name
is an empty string.

59085 [ENOTDIR]

59086 A component of the path prefix of the pathname in the socket address is not a
59087 directory, or the pathname in the socket address contains at least one
59088 non-*<slash>* character and ends with one or more trailing *<slash>* characters
59089 and the last pathname component names an existing file that is neither a
directory nor a symbolic link to a directory.

59090 The *sendmsg()* function may fail if:

59091 [EACCES] Search permission is denied for a component of the path prefix; or write access
59092 to the named socket is denied.

59093	[EDESTADDRREQ]	
59094		The socket is not connection-mode and does not have its peer address set, and
59095		no destination address was specified.
59096	[EHOSTUNREACH]	
59097		The destination host cannot be reached (probably because the host is down or
59098		a remote router cannot reach it).
59099	[EIO]	An I/O error occurred while reading from or writing to the file system.
59100	[EISCONN]	A destination address was specified and the socket is already connected.
59101	[ENETDOWN]	The local network interface used to reach the destination is down.
59102	[ENETUNREACH]	
59103		No route to the network is present.
59104	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
59105	[ENOMEM]	Insufficient memory was available to fulfill the request.
59106	If the address family of the socket is AF_UNIX, then <i>sendmsg()</i> may fail if:	
59107	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
59108		resolution of the pathname in the socket address.
59109	[ENAMETOOLONG]	
59110		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
59111		symbolic link produced an intermediate result with a length that exceeds
59112		{PATH_MAX}.

EXAMPLES

59113 Done.

APPLICATION USAGE

59116 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.

RATIONALE

59118 None.

FUTURE DIRECTIONS

59120 None.

SEE ALSO

59122 *getsockopt()*, *poll()*, *pselect()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendto()*, *setsockopt()*,
59123 *shutdown()*, *socket()*

59124 XBD <sys/socket.h>

CHANGE HISTORY

59126 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

59127 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
59128 [ELOOP] error condition is added.

Issue 7

59130 Austin Group Interpretation 1003.1-2001 #073 is applied, updating the DESCRIPTION.

59131 Austin Group Interpretation 1003.1-2001 #143 is applied.

59132 The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended
59133 API Set Part 2.

sendmsg()

System Interfaces

59134

The [EPIPE] error is modified.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

sendto()**NAME**

sendto — send a message on a socket

SYNOPSIS

```
#include <sys/socket.h>
```

```
ssize_t sendto(int socket, const void *message, size_t length,
               int flags, const struct sockaddr *dest_addr,
               socklen_t dest_len);
```

DESCRIPTION

The *sendto()* function shall send a message through a connection-mode or connectionless-mode socket.

If the socket is a connectionless-mode socket, the message shall be sent to the address specified by *dest_addr* if no pre-specified peer address has been set. If a peer address has been pre-specified, either the message shall be sent to the address specified by *dest_addr* (overriding the pre-specified peer address), or the function shall return -1 and set *errno* to [EISCONN].

If the socket is connection-mode, *dest_addr* shall be ignored.

The *sendto()* function takes the following arguments:

<i>socket</i>	Specifies the socket file descriptor.
<i>message</i>	Points to a buffer containing the message to be sent.
<i>length</i>	Specifies the size of the message in bytes.
<i>flags</i>	Specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more of the following flags:
MSG_EOR	Terminates a record (if supported by the protocol).
MSG_OOB	Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
MSG_NOSIGNAL	Requests not to send the SIGPIPE signal if an attempt to send is made on a stream-oriented socket that is no longer connected. The [EPIPE] error shall still be returned.
<i>dest_addr</i>	Points to a sockaddr structure containing the destination address. The length and format of the address depend on the address family of the socket.
<i>dest_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>dest_addr</i> argument.

If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, *sendto()* shall fail if the SO_BROADCAST option is not set for the socket.

The *dest_addr* argument specifies the address of the target.

The *length* argument specifies the length of the message.

Successful completion of a call to *sendto()* does not guarantee delivery of the message. A return value of -1 indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have O_NONBLOCK set, *sendto()* shall block until space is available. If space is not available at the sending socket to hold the message to be transmitted

sendto()*System Interfaces*

- 59177 and the socket file descriptor does have O_NONBLOCK set, *sendto()* shall fail.
- 59178 The socket in use may require the process to have appropriate privileges to use the *sendto()*
- 59179 function.
- 59180 **RETURN VALUE**
- 59181 Upon successful completion, *sendto()* shall return the number of bytes sent. Otherwise, -1 shall
- 59182 be returned and *errno* set to indicate the error.
- 59183 **ERRORS**
- 59184 The *sendto()* function shall fail if:
- 59185 [EAFNOSUPPORT]
- 59186 Addresses in the specified address family cannot be used with this socket.
- 59187 [EAGAIN] or [EWOULDBLOCK]
- 59188 The socket's file descriptor is marked O_NONBLOCK and the requested
- 59189 operation would block.
- 59190 [EBADF] The *socket* argument is not a valid file descriptor.
- 59191 [ECONNRESET] A connection was forcibly closed by a peer.
- 59192 [EINTR] A signal interrupted *sendto()* before any data was transmitted.
- 59193 [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.
- 59194 [ENOTCONN] The socket is connection-mode but is not connected.
- 59195 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 59196 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or
- 59197 more of the values set in *flags*.
- 59198 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is
- 59199 no longer connected. In the latter case, and if the socket is of type
- 59200 SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not
- 59201 set, the SIGPIPE signal is generated to the calling thread.
- 59202 If the address family of the socket is AF_UNIX, then *sendto()* shall fail if:
- 59203 [EIO] An I/O error occurred while reading from or writing to the file system.
- 59204 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
- 59205 in the socket address.
- 59206 [ENAMETOOLONG]
- 59207 The length of a component of a pathname is longer than {NAME_MAX}.
- 59208 [ENOENT] A component of the pathname does not name an existing file or the pathname
- 59209 is an empty string.
- 59210 [ENOTDIR] A component of the path prefix of the pathname in the socket address is not a
- 59211 directory, or the pathname in the socket address contains at least one
- 59212 non-*<slash>* character and ends with one or more trailing *<slash>* characters
- 59213 and the last pathname component names an existing file that is neither a
- 59214 directory nor a symbolic link to a directory.

59215	The <i>sendto()</i> function may fail if:	
59216	[EACCES]	Search permission is denied for a component of the path prefix; or write access to the named socket is denied.
59217		
59218	[EDESTADDRREQ]	
59219		The socket is not connection-mode and does not have its peer address set, and no destination address was specified.
59220		
59221	[EHOSTUNREACH]	
59222		The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).
59223		
59224	[EINVAL]	The <i>dest_len</i> argument is not a valid length for the address family.
59225	[EIO]	An I/O error occurred while reading from or writing to the file system.
59226	[EISCONN]	A destination address was specified and the socket is already connected.
59227	[ENETDOWN]	The local network interface used to reach the destination is down.
59228	[ENETUNREACH]	
59229		No route to the network is present.
59230	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
59231	[ENOMEM]	Insufficient memory was available to fulfill the request.
59232	If the address family of the socket is AF_UNIX, then <i>sendto()</i> may fail if:	
59233	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in the socket address.
59234		
59235	[ENAMETOOLONG]	
59236		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
59237		
59238		
59239	EXAMPLES	
59240	None.	
59241	APPLICATION USAGE	
59242	The <i>select()</i> and <i>poll()</i> functions can be used to determine when it is possible to send more data.	
59243	RATIONALE	
59244	None.	
59245	FUTURE DIRECTIONS	
59246	None.	
59247	SEE ALSO	
59248	<i>getsockopt()</i> , <i>poll()</i> , <i>pselect()</i> , <i>recv()</i> , <i>recvfrom()</i> , <i>recvmsg()</i> , <i>send()</i> , <i>sendmsg()</i> , <i>setsockopt()</i> , <i>shutdown()</i> , <i>socket()</i>	
59249		
59250	XBD <sys/socket.h>	
59251	CHANGE HISTORY	
59252	First released in Issue 6. Derived from the XNS, Issue 5.2 specification.	
59253	The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.	
59254		

sendto()*System Interfaces***Issue 7**

- 59255 Austin Group Interpretations 1003.1-2001 #035 and #073 are applied, updating the [EISCONN]
59256 error and the DESCRIPTION.
59257
- 59258 Austin Group Interpretation 1003.1-2001 #143 is applied, clarifying the [ENAMETOOLONG]
59259 error condition.
- 59260 The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended
59261 API Set Part 2.
- 59262 The [EPIPE] error is modified.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

59263 **NAME**

59264 setbuf — assign buffering to a stream

59265 **SYNOPSIS**

59266 #include <stdio.h>

59267 void setbuf(FILE *restrict stream, char *restrict buf);

59268 **DESCRIPTION**

59269 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 59270 conflict between the requirements described here and the ISO C standard is unintentional. This
 59271 volume of POSIX.1-2008 defers to the ISO C standard.

59272 Except that it returns no value, the function call:

59273 setbuf(stream, buf)

59274 shall be equivalent to:

59275 setvbuf(stream, buf, _IOFBF, BUFSIZ)

59276 if *buf* is not a null pointer, or to:

59277 setvbuf(stream, buf, _IONBF, BUFSIZ)

59278 if *buf* is a null pointer.59279 **RETURN VALUE**59280 The *setbuf()* function shall not return a value.59281 **ERRORS**

59282 No errors are defined.

59283 **EXAMPLES**

59284 None.

59285 **APPLICATION USAGE**

59286 A common source of error is allocating buffer space as an “automatic” variable in a code block,
 59287 and then failing to close the stream in the same block.

59288 With *setbuf()*, allocating a buffer of BUFSIZ bytes does not necessarily imply that all of BUFSIZ
 59289 bytes are used for the buffer area.

59290 **RATIONALE**

59291 None.

59292 **FUTURE DIRECTIONS**

59293 None.

59294 **SEE ALSO**59295 *fclose()*, *setvbuf()*

59296 XBD <stdio.h>

59297 **CHANGE HISTORY**

59298 First released in Issue 1. Derived from Issue 1 of the SVID.

59299 **Issue 6**59300 The prototype for *setbuf()* is updated for alignment with the ISO/IEC 9899:1999 standard.

setegid()*System Interfaces*59301 **NAME**

59302 setegid — set the effective group ID

59303 **SYNOPSIS**

59304 #include <unistd.h>

59305 int setegid(gid_t gid);

59306 **DESCRIPTION**

59307 If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate
 59308 privileges, *setegid()* shall set the effective group ID of the calling process to *gid*; the real group
 59309 ID, saved set-group-ID, and any supplementary group IDs shall remain unchanged.

59310 The *setegid()* function shall not affect the supplementary group list in any way.

59311 **RETURN VALUE**

59312 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 59313 indicate the error.

59314 **ERRORS**

59315 The *setegid()* function shall fail if:

59316 [EINVAL] The value of the *gid* argument is invalid and is not supported by the
 59317 implementation.

59318 [EPERM] The process does not have appropriate privileges and *gid* does not match the
 59319 real group ID or the saved set-group-ID.

59320 **EXAMPLES**

59321 None.

59322 **APPLICATION USAGE**

59323 None.

59324 **RATIONALE**59325 Refer to the RATIONALE section in *setuid()*.59326 **FUTURE DIRECTIONS**

59327 None.

59328 **SEE ALSO**59329 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*

59330 XBD <sys/types.h>, <unistd.h>

59331 **CHANGE HISTORY**

59332 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

NAME

setenv — add or change environment variable

SYNOPSIS

```
#include <stdlib.h>

int setenv(const char *envname, const char *envval, int overwrite);
```

DESCRIPTION

The *setenv()* function shall update or add a variable in the environment of the calling process. The *envname* argument points to a string containing the name of an environment variable to be added or altered. The environment variable shall be set to the value to which *envval* points. The function shall fail if *envname* points to a string which contains an '=' character. If the environment variable named by *envname* already exists and the value of *overwrite* is non-zero, the function shall return success and the environment shall be updated. If the environment variable named by *envname* already exists and the value of *overwrite* is zero, the function shall return success and the environment shall remain unchanged.

If the application modifies *environ* or the pointers to which it points, the behavior of *setenv()* is undefined. The *setenv()* function shall update the list of pointers to which *environ* points.

The strings described by *envname* and *envval* are copied by this function.

The *setenv()* function need not be thread-safe.

RETURN VALUE

Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to indicate the error, and the environment shall be unchanged.

ERRORS

The *setenv()* function shall fail if:

[EINVAL]	The <i>name</i> argument is a null pointer, points to an empty string, or points to a string containing an '=' character.
[ENOMEM]	Insufficient memory was available to add a variable or its value to the environment.

EXAMPLES

None.

APPLICATION USAGE

See *exec()* for restrictions on changing the environment in multi-threaded applications.

RATIONALE

Unanticipated results may occur if *setenv()* changes the external variable *environ*. In particular, if the optional *envp* argument to *main()* is present, it is not changed, and thus may point to an obsolete copy of the environment (as may any other copy of *environ*). However, other than the aforementioned restriction, the standard developers intended that the traditional method of walking through the environment by way of the *environ* pointer must be supported.

It was decided that *setenv()* should be required by this version because it addresses a piece of missing functionality, and does not impose a significant burden on the implementor.

There was considerable debate as to whether the System V *putenv()* function or the BSD *setenv()* function should be required as a mandatory function. The *setenv()* function was chosen because it permitted the implementation of the *unsetenv()* function to delete environmental variables, without specifying an additional interface. The *putenv()* function is available as part of the XSI option.

setenv()*System Interfaces*

59377 The standard developers considered requiring that *setenv()* indicate an error when a call to it
59378 would result in exceeding {ARG_MAX}. The requirement was rejected since the condition might
59379 be temporary, with the application eventually reducing the environment size. The ultimate
59380 success or failure depends on the size at the time of a call to *exec*, which returns an indication of
59381 this error condition.

FUTURE DIRECTIONS

59382 None.
59383

SEE ALSO

59384 *exec*, *getenv()*, *unsetenv()*
59385

59386 XBD *<stdlib.h>*, *<sys/types.h>*, *<unistd.h>*

CHANGE HISTORY

59387 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.
59388

59389 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/55 is applied, adding references to *exec* in
59390 the APPLICATION USAGE and SEE ALSO sections.

Issue 7

59391 Austin Group Interpretation 1003.1-2001 #156 is applied.
59392

System Interfaces

seteuid()**NAME**

seteuid — set effective user ID

SYNOPSIS

```
#include <unistd.h>

int seteuid(uid_t uid);
```

DESCRIPTION

If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate privileges, *seteuid()* shall set the effective user ID of the calling process to *uid*; the real user ID and saved set-user-ID shall remain unchanged.

The *seteuid()* function shall not affect the supplementary group list in any way.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *seteuid()* function shall fail if:

[EINVAL] The value of the *uid* argument is invalid and is not supported by the implementation.

[EPERM] The process does not have appropriate privileges and *uid* does not match the real user ID or the saved set-user-ID.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to the RATIONALE section in *setuid()*.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*

XBD <sys/types.h>, <unistd.h>

CHANGE HISTORY

First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/123 is applied, making an editorial correction to the [EPERM] error in the ERRORS section.

setgid()*System Interfaces*59427 **NAME**

59428 setgid — set-group-ID

59429 **SYNOPSIS**

59430 #include <unistd.h>

59431 int setgid(gid_t gid);

59432 **DESCRIPTION**59433 If the process has appropriate privileges, *setgid()* shall set the real group ID, effective group ID,
59434 and the saved set-group-ID of the calling process to *gid*.59435 If the process does not have appropriate privileges, but *gid* is equal to the real group ID or the
59436 saved set-group-ID, *setgid()* shall set the effective group ID to *gid*; the real group ID and saved
59437 set-group-ID shall remain unchanged.59438 The *setgid()* function shall not affect the supplementary group list in any way.

59439 Any supplementary group IDs of the calling process shall remain unchanged.

59440 **RETURN VALUE**59441 Upon successful completion, 0 is returned. Otherwise, -1 shall be returned and *errno* set to
59442 indicate the error.59443 **ERRORS**59444 The *setgid()* function shall fail if:59445 [EINVAL] The value of the *gid* argument is invalid and is not supported by the
59446 implementation.59447 [EPERM] The process does not have appropriate privileges and *gid* does not match the
59448 real group ID or the saved set-group-ID.59449 **EXAMPLES**

59450 None.

59451 **APPLICATION USAGE**

59452 None.

59453 **RATIONALE**59454 Refer to the RATIONALE section in *setuid()*.59455 **FUTURE DIRECTIONS**

59456 None.

59457 **SEE ALSO**59458 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setregid()*, *setreuid()*, *setuid()*

59459 XBD <sys/types.h>, <unistd.h>

59460 **CHANGE HISTORY**

59461 First released in Issue 1. Derived from Issue 1 of the SVID.

59462 **Issue 6**

59463 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

59464 The following new requirements on POSIX implementations derive from alignment with the
59465 Single UNIX Specification:

- 59466 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
59467 required for conforming implementations of previous POSIX specifications, it was not
59468 required for UNIX applications.

- 59469 • Functionality associated with `_POSIX_SAVED_IDS` is now mandated. This is a FIPS
59470 requirement.

59471 The following changes were made to align with the IEEE P1003.1a draft standard:

- 59472 • The effects of `setgid()` in processes without appropriate privileges are changed.
59473 • A requirement that the supplementary group list is not affected is added.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

setgrent()*System Interfaces*59474 **NAME**

59475 setgrent — reset the group database to the first entry

59476 **SYNOPSIS**

```
59477 XSI     #include <grp.h>  
59478       void setgrent(void);
```

59479 **DESCRIPTION**59480 Refer to *endgrent()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***sethostent()**59481 **NAME**

59482 sethostent — network host database functions

59483 **SYNOPSIS**

59484 #include <netdb.h>

59485 void sethostent(int *stayopen*);59486 **DESCRIPTION**59487 Refer to *endhostent()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

setitimer()*System Interfaces*59488 **NAME**

59489 setitimer — set the value of an interval timer

59490 **SYNOPSIS**

59491 OB XSI #include <sys/time.h>

```
59492       int setitimer(int which, const struct itimerval *restrict value,  
59493                    struct itimerval *restrict ovalue);
```

59494 **DESCRIPTION**59495 Refer to *getitimer()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

setjmp — set jump point for a non-local goto

SYNOPSIS

```
#include <setjmp.h>

int setjmp(jmp_buf env);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

A call to *setjmp()* shall save the calling environment in its *env* argument for later use by *longjmp()*.

It is unspecified whether *setjmp()* is a macro or a function. If a macro definition is suppressed in order to access an actual function, or a program defines an external identifier with the name *setjmp*, the behavior is undefined.

An application shall ensure that an invocation of *setjmp()* appears in one of the following contexts only:

- The entire controlling expression of a selection or iteration statement
- One operand of a relational or equality operator with the other operand an integral constant expression, with the resulting expression being the entire controlling expression of a selection or iteration statement
- The operand of a unary '!' operator with the resulting expression being the entire controlling expression of a selection or iteration
- The entire expression of an expression statement (possibly cast to **void**)

If the invocation appears in any other context, the behavior is undefined.

RETURN VALUE

If the return is from a direct invocation, *setjmp()* shall return 0. If the return is from a call to *longjmp()*, *setjmp()* shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

In general, *sigsetjmp()* is more useful in dealing with errors and interrupts encountered in a low-level subroutine of a program.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

longjmp(), *sigsetjmp()*

XBD <setjmp.h>

setjmp()*System Interfaces***CHANGE HISTORY**

59537
59538 First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

59539
59540 The normative text is updated to avoid use of the term “must” for application requirements.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

setkey()

59541 NAME

59542 setkey — set encoding key (CRYPT)

59543 SYNOPSIS

```
59544 XSI      #include <stdlib.h>
59545          void setkey(const char *key);
```

59546 DESCRIPTION

59547 The *setkey()* function provides access to an implementation-defined encoding algorithm. The
 59548 argument of *setkey()* is an array of length 64 bytes containing only the bytes with numerical
 59549 value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is
 59550 ignored; this gives a 56-bit key which is used by the algorithm. This is the key that shall be used
 59551 with the algorithm to encode a string *block* passed to *encrypt()*.

59552 The *setkey()* function shall not change the setting of *errno* if successful. An application wishing to
 59553 check for error situations should set *errno* to 0 before calling *setkey()*. If *errno* is non-zero on
 59554 return, an error has occurred.

59555 The *setkey()* function need not be thread-safe.

59556 RETURN VALUE

59557 No values are returned.

59558 ERRORS

59559 The *setkey()* function shall fail if:

59560 [ENOSYS] The functionality is not supported on this implementation.

59561 EXAMPLES

59562 None.

59563 APPLICATION USAGE

59564 Decoding need not be implemented in all environments. This is related to government
 59565 restrictions in some countries on encryption and decryption routines. Historical practice has
 59566 been to ship a different version of the encryption library without the decryption feature in the
 59567 routines supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

59568 RATIONALE

59569 None.

59570 FUTURE DIRECTIONS

59571 None.

59572 SEE ALSO

59573 *crypt()*, *encrypt()*

59574 XBD <stdlib.h>

59575 CHANGE HISTORY

59576 First released in Issue 1. Derived from Issue 1 of the SVID.

59577 Issue 5

59578 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

59579 Issue 7

59580 Austin Group Interpretation 1003.1-2001 #156 is applied.

setlocale()

System Interfaces

59581 **NAME**

59582 setlocale — set program locale

59583 **SYNOPSIS**

59584 #include <locale.h>

59585 char *setlocale(int *category*, const char **locale*);59586 **DESCRIPTION**

59587 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 59588 conflict between the requirements described here and the ISO C standard is unintentional. This
 59589 volume of POSIX.1-2008 defers to the ISO C standard.

59590 The *setlocale()* function selects the appropriate piece of the locale of the process, as specified by
 59591 the *category* and *locale* arguments, and may be used to change or query the entire locale of the
 59592 process or portions thereof. The value *LC_ALL* for *category* names the entire locale of the process;
 59593 other values for *category* name only a part of the locale of the process:

59594 *LC_COLLATE* Affects the behavior of regular expressions and the collation functions.

59595 *LC_CTYPE* Affects the behavior of regular expressions, character classification, character
 59596 conversion functions, and wide-character functions.

59597 CX *LC_MESSAGES* Affects what strings are expected by commands and utilities as affirmative or
 59598 negative responses.

59599 XSI It also affects what strings are given by commands and utilities as affirmative
 59600 or negative responses, and the content of messages.

59601 *LC_MONETARY* Affects the behavior of functions that handle monetary values.

59602 *LC_NUMERIC* Affects the behavior of functions that handle numeric values.

59603 *LC_TIME* Affects the behavior of the time conversion functions.

59604 The *locale* argument is a pointer to a character string containing the required setting of *category*.
 59605 The contents of this string are implementation-defined. In addition, the following preset values
 59606 of *locale* are defined for all settings of *category*:

59607 CX "POSIX" Specifies the minimal environment for C-language translation called the
 59608 POSIX locale. If *setlocale()* is not invoked, the POSIX locale is the default at
 59609 entry to *main()*.

59610 "C" Equivalent to "POSIX".

59611 CX "" Specifies an implementation-defined native environment. The determination
 59612 of the name of the new locale for the specified category depends on the value
 59613 of the associated environment variables, *LC_** and *LANG*; see XBD Chapter 7
 59614 (on page 135) and Chapter 8 (on page 173).

59615 A null pointer Used to direct *setlocale()* to query the current internationalized environment
 59616 and return the name of the locale.

59617 CX Setting all of the categories of the locale of the process is similar to successively setting each
 59618 individual category of the locale of the process, except that all error checking is done before any
 59619 actions are performed. To set all the categories of the locale of the process, *setlocale()* is invoked
 59620 as:

59621 setlocale(LC_ALL, "");

59622 In this case, *setlocale()* shall first verify that the values of all the environment variables it needs
 59623 according to the precedence rules (described in XBD Chapter 8, on page 173) indicate supported

locales. If the value of any of these environment variable searches yields a locale that is not supported (and non-null), *setlocale()* shall return a null pointer and the locale of the process shall not be changed. If all environment variables name supported locales, *setlocale()* shall proceed as if it had been called for each category, using the appropriate value from the associated environment variable or from the implementation-defined default if there is no such value.

The locale state is common to all threads within a process.

RETURN VALUE

Upon successful completion, *setlocale()* shall return the string associated with the specified category for the new locale. Otherwise, *setlocale()* shall return a null pointer and the locale of the process is not changed.

A null pointer for *locale* causes *setlocale()* to return a pointer to the string associated with the category for the current locale of the process. The locale of the process shall not be changed.

The string returned by *setlocale()* is such that a subsequent call with that string and its associated category shall restore that part of the locale of the process. The application shall not modify the string returned which may be overwritten by a subsequent call to *setlocale()*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The following code illustrates how a program can initialize the international environment for one language, while selectively modifying the locale of the process such that regular expressions and string operations can be applied to text recorded in a different language:

```
setlocale(LC_ALL, "De");
setlocale(LC_COLLATE, "Fr@dict");
```

Internationalized programs must call *setlocale()* to initiate a specific language operation. This can be done by calling *setlocale()* as follows:

```
setlocale(LC_ALL, "");
```

Changing the setting of *LC_MESSAGES* has no effect on catalogs that have already been opened by calls to *catopen()*.

RATIONALE

The ISO C standard defines a collection of functions to support internationalization. One of the most significant aspects of these functions is a facility to set and query the *international environment*. The international environment is a repository of information that affects the behavior of certain functionality, namely:

1. Character handling
2. Collating
3. Date/time formatting
4. Numeric editing
5. Monetary formatting
6. Messaging

The *setlocale()* function provides the application developer with the ability to set all or portions,

setlocale()*System Interfaces*

called *categories*, of the international environment. These categories correspond to the areas of functionality mentioned above. The syntax for *setlocale()* is as follows:

```
char *setlocale(int category, const char *locale);
```

where *category* is the name of one of following categories, namely:

```
LC_COLLATE
LC_CTYPE
LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME
```

In addition, a special value called *LC_ALL* directs *setlocale()* to set all categories.

There are two primary uses of *setlocale()*:

1. Querying the international environment to find out what it is set to
2. Setting the international environment, or *locale*, to a specific value

The behavior of *setlocale()* in these two areas is described below. Since it is difficult to describe the behavior in words, examples are used to illustrate the behavior of specific uses.

To query the international environment, *setlocale()* is invoked with a specific category and the null pointer as the locale. The null pointer is a special directive to *setlocale()* that tells it to query rather than set the international environment. The following syntax is used to query the name of the international environment:

```
setlocale({LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, \
          LC_NUMERIC, LC_TIME}, (char *) NULL);
```

The *setlocale()* function shall return the string corresponding to the current international environment. This value may be used by a subsequent call to *setlocale()* to reset the international environment to this value. However, it should be noted that the return value from *setlocale()* may be a pointer to a static area within the function and is not guaranteed to remain unchanged (that is, it may be modified by a subsequent call to *setlocale()*). Therefore, if the purpose of calling *setlocale()* is to save the value of the current international environment so it can be changed and reset later, the return value should be copied to an array of **char** in the calling program.

There are three ways to set the international environment with *setlocale()*:

```
setlocale(category, string)
```

This usage sets a specific *category* in the international environment to a specific value corresponding to the value of the *string*. A specific example is provided below:

```
setlocale(LC_ALL, "fr_FR.ISO-8859-1");
```

In this example, all categories of the international environment are set to the locale corresponding to the string "fr_FR.ISO-8859-1", or to the French language as spoken in France using the ISO/IEC 8859-1:1998 standard codeset.

If the string does not correspond to a valid locale, *setlocale()* shall return a null pointer and the international environment is not changed. Otherwise, *setlocale()* shall return the name of the locale just set.

59707 `setlocale(category, "C")`

59708 The ISO C standard states that one locale must exist on all conforming implementations.
 59709 The name of the locale is C and corresponds to a minimal international environment needed
 59710 to support the C programming language.

59711 `setlocale(category, "")`

59712 This sets a specific category to an implementation-defined default. This corresponds to the
 59713 value of the environment variables.

59714 FUTURE DIRECTIONS

59715 None.

59716 SEE ALSO

59717 `exec`, `fprintf()`, `fscanf()`, `isalnum()`, `isalpha()`, `isblank()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`,
 59718 `isprint()`, `ispunct()`, `isspace()`, `isupper()`, `iswalnum()`, `iswalpha()`, `iswblank()`, `iswcntrl()`, `iswctype()`,
 59719 `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`,
 59720 `isxdigit()`, `localeconv()`, `mblen()`, `mbstowcs()`, `mbtowc()`, `nl_langinfo()`, `setlocale()`, `strcoll()`,
 59721 `strerror()`, `strfmon()`, `strsignal()`, `strtod()`, `strxfrm()`, `tolower()`, `toupper()`, `towlower()`, `towupper()`,
 59722 `uselocale()`, `wcscoll()`, `wctod()`, `wcstombs()`, `wcsxfrm()`, `wctomb()`

59723 XBD Chapter 7 (on page 135), Chapter 8 (on page 173), `<langinfo.h>`, `<locale.h>`

59724 CHANGE HISTORY

59725 First released in Issue 3.

59726 Issue 5

59727 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

59728 Issue 6

59729 Extensions beyond the ISO C standard are marked.

59730 The normative text is updated to avoid use of the term “must” for application requirements.

59731 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/124 is applied, updating the
 59732 DESCRIPTION to clarify the behavior of:

59733 `setlocale(LC_ALL, "");`

59734 Issue 7

59735 Functionality relating to the Threads option is moved to the Base.

setlogmask()*System Interfaces*59736 **NAME**

59737 setlogmask — set the log priority mask

59738 **SYNOPSIS**59739 XSI `#include <syslog.h>`59740 `int setlogmask(int maskpri);`59741 **DESCRIPTION**59742 Refer to *closelog()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***setnetent()**59743 **NAME**

59744 setnetent — network database function

59745 **SYNOPSIS**

59746 #include <netdb.h>

59747 void setnetent(int stayopen);

59748 **DESCRIPTION**59749 Refer to *endnetent()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

setpgid()*System Interfaces*59750 **NAME**59751 `setpgid` — set process group ID for job control59752 **SYNOPSIS**59753 `#include <unistd.h>`59754 `int setpgid(pid_t pid, pid_t pgid);`59755 **DESCRIPTION**59756 The `setpgid()` function shall either join an existing process group or create a new process group
59757 within the session of the calling process.

59758 The process group ID of a session leader shall not change.

59759 Upon successful completion, the process group ID of the process with a process ID that matches
59760 `pid` shall be set to `pgid`.59761 As a special case, if `pid` is 0, the process ID of the calling process shall be used. Also, if `pgid` is 0,
59762 the process ID of the indicated process shall be used.59763 **RETURN VALUE**59764 Upon successful completion, `setpgid()` shall return 0; otherwise, `-1` shall be returned and `errno`
59765 shall be set to indicate the error.59766 **ERRORS**59767 The `setpgid()` function shall fail if:59768 [EACCES] The value of the `pid` argument matches the process ID of a child process of the
59769 calling process and the child process has successfully executed one of the `exec`
59770 functions.59771 [EINVAL] The value of the `pgid` argument is less than 0, or is not a value supported by
59772 the implementation.59773 [EPERM] The process indicated by the `pid` argument is a session leader.59774 [EPERM] The value of the `pid` argument matches the process ID of a child process of the
59775 calling process and the child process is not in the same session as the calling
59776 process.59777 [EPERM] The value of the `pgid` argument is valid but does not match the process ID of
59778 the process indicated by the `pid` argument and there is no process with a
59779 process group ID that matches the value of the `pgid` argument in the same
59780 session as the calling process.59781 [ESRCH] The value of the `pid` argument does not match the process ID of the calling
59782 process or of a child process of the calling process.59783 **EXAMPLES**

59784 None.

59785 **APPLICATION USAGE**

59786 None.

59787 **RATIONALE**59788 The `setpgid()` function shall group processes together for the purpose of signaling, placement in
59789 foreground or background, and other job control actions.59790 The `setpgid()` function is similar to the `setpgrp()` function of 4.2 BSD, except that 4.2 BSD allowed
59791 the specified new process group to assume any value. This presents certain security problems
59792 and is more flexible than necessary to support job control.

To provide tighter security, *setpgid()* only allows the calling process to join a process group already in use inside its session or create a new process group whose process group ID was equal to its process ID.

When a job control shell spawns a new job, the processes in the job must be placed into a new process group via *setpgid()*. There are two timing constraints involved in this action:

1. The new process must be placed in the new process group before the appropriate program is launched via one of the *exec* functions.
2. The new process must be placed in the new process group before the shell can correctly send signals to the new process group.

To address these constraints, the following actions are performed. The new processes call *setpgid()* to alter their own process groups after *fork()* but before *exec*. This satisfies the first constraint. Under 4.3 BSD, the second constraint is satisfied by the synchronization property of *vfork()*; that is, the shell is suspended until the child has completed the *exec*, thus ensuring that the child has completed the *setpgid()*. A new version of *fork()* with this same synchronization property was considered, but it was decided instead to merely allow the parent shell process to adjust the process group of its child processes via *setpgid()*. Both timing constraints are now satisfied by having both the parent shell and the child attempt to adjust the process group of the child process; it does not matter which succeeds first.

Since it would be confusing to an application to have its process group change after it began executing (that is, after *exec*), and because the child process would already have adjusted its process group before this, the [EACCES] error was added to disallow this.

One non-obvious use of *setpgid()* is to allow a job control shell to return itself to its original process group (the one in effect when the job control shell was executed). A job control shell does this before returning control back to its parent when it is terminating or suspending itself as a way of restoring its job control “state” back to what its parent would expect. (Note that the original process group of the job control shell typically matches the process group of its parent, but this is not necessarily always the case.)

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getpgrp()*, *setsid()*, *tcsetpgrp()*

XBD [`<sys/types.h>`](#), [`<unistd.h>`](#)

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the SYNOPSIS, the optional include of the [`<sys/types.h>`](#) header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include [`<sys/types.h>`](#) has been removed. Although [`<sys/types.h>`](#) was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The *setpgid()* function is mandatory since `_POSIX_JOB_CONTROL` is required to be defined in this version. This is a FIPS requirement.

setpgid()*System Interfaces*

59836 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/56 is applied, changing the wording in
59837 the DESCRIPTION from “the process group ID of the indicated process shall be used” to “the
59838 process ID of the indicated process shall be used”. This change reverts the wording to as in the
59839 ISO POSIX-1: 1996 standard; it appeared to be an unintentional change.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

setpgrp()59840 **NAME**

59841 setpgrp — set the process group ID

59842 **SYNOPSIS**

```
59843 OB XSI  #include <unistd.h>
59844         pid_t setpgrp(void);
```

59845 **DESCRIPTION**

59846 If the calling process is not already a session leader, *setpgrp()* sets the process group ID of the
 59847 calling process to the process ID of the calling process. If *setpgrp()* creates a new session, then the
 59848 new session has no controlling terminal.

59849 The *setpgrp()* function has no effect when the calling process is a session leader.

59850 **RETURN VALUE**

59851 Upon completion, *setpgrp()* shall return the process group ID.

59852 **ERRORS**

59853 No errors are defined.

59854 **EXAMPLES**

59855 None.

59856 **APPLICATION USAGE**

59857 It is unspecified whether this function behaves as *setpgid(0,0)* or *setsid()* unless the process is
 59858 already a session leader. Therefore, applications are encouraged to use *setpgid()* or *setsid()* as
 59859 appropriate.

59860 **RATIONALE**

59861 None.

59862 **FUTURE DIRECTIONS**

59863 The *setpgrp()* function may be removed in a future version.

59864 **SEE ALSO**

59865 *exec*, *fork()*, *getpid()*, *getsid()*, *kill()*, *setpgid()*, *setsid()*

59866 XBD *<unistd.h>*

59867 **CHANGE HISTORY**

59868 First released in Issue 4, Version 2.

59869 **Issue 5**

59870 Moved from X/OPEN UNIX extension to BASE.

59871 **Issue 7**

59872 The *setpgrp()* function is marked obsolescent.

setpriority()*System Interfaces*59873 **NAME**

59874 setpriority — set the nice value

59875 **SYNOPSIS**

```
59876 XSI      #include <sys/resource.h>  
59877          int setpriority(int which, id_t who, int nice);
```

59878 **DESCRIPTION**59879 Refer to *getpriority()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***setprotoent()**59880 **NAME**

59881 setprotoent — network protocol database functions

59882 **SYNOPSIS**

59883 #include <netdb.h>

59884 void setprotoent(int *stayopen*);59885 **DESCRIPTION**59886 Refer to *endprotoent()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

setpwent()*System Interfaces*59887 **NAME**

59888 setpwent — user database function

59889 **SYNOPSIS**

59890 XSI #include <pwd.h>

59891 void setpwent(void);

59892 **DESCRIPTION**59893 Refer to *endpwent()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

setregid — set real and effective group IDs

SYNOPSIS

```
#include <unistd.h>

int setregid(gid_t rgid, gid_t egid);
```

DESCRIPTION

The *setregid()* function shall set the real and effective group IDs of the calling process.

If *rgid* is -1 , the real group ID shall not be changed; if *egid* is -1 , the effective group ID shall not be changed.

The real and effective group IDs may be set to different values in the same call.

Only a process with appropriate privileges can set the real group ID and the effective group ID to any valid value.

A non-privileged process can set either the real group ID to the saved set-group-ID from one of the *exec* family of functions, or the effective group ID to the saved set-group-ID or the real group ID.

If the real group ID is being set (*rgid* is not -1), or the effective group ID is being set to a value not equal to the real group ID, then the saved set-group-ID of the current process shall be set equal to the new effective group ID.

Any supplementary group IDs of the calling process remain unchanged.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error, and neither of the group IDs are changed.

ERRORS

The *setregid()* function shall fail if:

[EINVAL] The value of the *rgid* or *egid* argument is invalid or out-of-range.

[EPERM] The process does not have appropriate privileges and a change other than changing the real group ID to the saved set-group-ID, or changing the effective group ID to the real group ID or the saved set-group-ID, was requested.

EXAMPLES

None.

APPLICATION USAGE

If a non-privileged set-group-ID process sets its effective group ID to its real group ID, it can only set its effective group ID back to the previous value if *rgid* was -1 in the *setregid()* call, since the saved-group-ID is not changed in that case. If *rgid* was equal to the real group ID in the *setregid()* call, then the saved set-group-ID will also have been changed to the real user ID.

RATIONALE

Earlier versions of this standard did not specify whether the saved set-group-ID was affected by *setregid()* calls. This version specifies common existing practice that constitutes an important security feature. The ability to set both the effective group ID and saved set-group-ID to be the same as the real group ID means that any security weakness in code that is executed after that point cannot result in malicious code being executed with the previous effective group ID. Privileged applications could already do this using just *setgid()*, but for non-privileged

setregid()

System Interfaces

59937 applications the only standard method available is to use this feature of *setregid()*.

59938 FUTURE DIRECTIONS

59939 None.

59940 SEE ALSO

59941 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setreuid()*, *setuid()*

59942 XBD <*unistd.h*>

59943 CHANGE HISTORY

59944 First released in Issue 4, Version 2.

59945 Issue 5

59946 Moved from X/OPEN UNIX extension to BASE.

59947 The DESCRIPTION is updated to indicate that the saved set-group-ID can be set by any of the
59948 *exec* family of functions, not just *execve()*.

59949 Issue 7

59950 SD5-XSH-ERN-177 is applied, adding the ability to set both the effective group ID and saved set-
59951 group-ID to be the same as the real group ID.

System Interfaces

setreuid()**NAME**

setreuid — set real and effective user IDs

SYNOPSIS

```
#include <unistd.h>

int setreuid(uid_t ruid, uid_t euid);
```

DESCRIPTION

The *setreuid()* function shall set the real and effective user IDs of the current process to the values specified by the *ruid* and *euid* arguments. If *ruid* or *euid* is -1 , the corresponding effective or real user ID of the current process shall be left unchanged.

A process with appropriate privileges can set either ID to any value. An unprivileged process can only set the effective user ID if the *euid* argument is equal to either the real, effective, or saved user ID of the process.

If the real user ID is being set (*ruid* is not -1), or the effective user ID is being set to a value not equal to the real user ID, then the saved set-user-ID of the current process shall be set equal to the new effective user ID.

It is unspecified whether a process without appropriate privileges is permitted to change the real user ID to match the current effective user ID or saved set-user-ID of the process.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setreuid()* function shall fail if:

[EINVAL]	The value of the <i>ruid</i> or <i>euid</i> argument is invalid or out-of-range.
[EPERM]	The current process does not have appropriate privileges, and either an attempt was made to change the effective user ID to a value other than the real user ID or the saved set-user-ID or an attempt was made to change the real user ID to a value not permitted by the implementation.

EXAMPLES**Setting the Effective User ID to the Real User ID**

The following example sets the effective user ID of the calling process to the real user ID, so that files created later will be owned by the current user. It also sets the saved set-user-ID to the real user ID, so any future attempt to set the effective user ID back to its previous value will fail.

```
#include <unistd.h>
#include <sys/types.h>
...
setreuid(getuid(), getuid());
...
```

APPLICATION USAGE

None.

setreuid()

System Interfaces

59991 **RATIONALE**

59992 Earlier versions of this standard did not specify whether the saved set-user-ID was affected by
59993 *setreuid()* calls. This version specifies common existing practice that constitutes an important
59994 security feature. The ability to set both the effective user ID and saved set-user-ID to be the same
59995 as the real user ID means that any security weakness in code that is executed after that point
59996 cannot result in malicious code being executed with the previous effective user ID. Privileged
59997 applications could already do this using just *setuid()*, but for non-privileged applications the
59998 only standard method available is to use this feature of *setreuid()*.

59999 **FUTURE DIRECTIONS**

60000 None.

60001 **SEE ALSO**

60002 *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setuid()*

60003 XBD [<unistd.h>](#)

60004 **CHANGE HISTORY**

60005 First released in Issue 4, Version 2.

60006 **Issue 5**

60007 Moved from X/OPEN UNIX extension to BASE.

60008 **Issue 7**

60009 SD5-XSH-ERN-177 is applied, adding the ability to set both the effective user ID and the saved
60010 set-user-ID to be the same as the real user ID.

*System Interfaces***setrlimit()**60011 **NAME**

60012 setrlimit — control maximum resource consumption

60013 **SYNOPSIS**

```
60014 XSI      #include <sys/resource.h>  
60015          int setrlimit(int resource, const struct rlimit *rlp);
```

60016 **DESCRIPTION**60017 Refer to *getrlimit()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

setservent()*System Interfaces*60018 **NAME**

60019 setservent — network services database functions

60020 **SYNOPSIS**

60021 #include <netdb.h>

60022 void setservent(int stayopen);

60023 **DESCRIPTION**60024 Refer to *endservent()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

setsid — create session and set process group ID

SYNOPSIS

```
#include <unistd.h>
```

```
pid_t setsid(void);
```

DESCRIPTION

The *setsid()* function shall create a new session, if the calling process is not a process group leader. Upon return the calling process shall be the session leader of this new session, shall be the process group leader of a new process group, and shall have no controlling terminal. The process group ID of the calling process shall be set equal to the process ID of the calling process. The calling process shall be the only process in the new process group and the only process in the new session.

RETURN VALUE

Upon successful completion, *setsid()* shall return the value of the new process group ID of the calling process. Otherwise, it shall return (**pid_t**)-1 and set *errno* to indicate the error.

ERRORS

The *setsid()* function shall fail if:

[EPERM]	The calling process is already a process group leader, or the process group ID of a process other than the calling process matches the process ID of the calling process.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *setsid()* function is similar to the *setpgp()* function of System V. System V, without job control, groups processes into process groups and creates new process groups via *setpgp()*; only one process group may be part of a login session.

Job control allows multiple process groups within a login session. In order to limit job control actions so that they can only affect processes in the same login session, this volume of POSIX.1-2008 adds the concept of a session that is created via *setsid()*. The *setsid()* function also creates the initial process group contained in the session. Additional process groups can be created via the *setpgid()* function. A System V process group would correspond to a POSIX System Interfaces session containing a single POSIX process group. Note that this function requires that the calling process not be a process group leader. The usual way to ensure this is true is to create a new process with *fork()* and have it call *setsid()*. The *fork()* function guarantees that the process ID of the new process does not match any existing process group ID.

FUTURE DIRECTIONS

None.

SEE ALSO

getsid(), *setpgid()*, *setpgp()*

XBD [<sys/types.h>](#), [<unistd.h>](#)

setsid()*System Interfaces*60067 **CHANGE HISTORY**

60068 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

60069 **Issue 6**60070 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.60071 The following new requirements on POSIX implementations derive from alignment with the
60072 Single UNIX Specification:

- 60073 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
60074 required for conforming implementations of previous POSIX specifications, it was not
60075 required for UNIX applications.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

setsockopt — set the socket options

SYNOPSIS

```
#include <sys/socket.h>
```

```
int setsockopt(int socket, int level, int option_name,
               const void *option_value, socklen_t option_len);
```

DESCRIPTION

The *setsockopt()* function shall set the option specified by the *option_name* argument, at the protocol level specified by the *level* argument, to the value pointed to by the *option_value* argument for the socket associated with the file descriptor specified by the *socket* argument.

The *level* argument specifies the protocol level at which the option resides. To set options at the socket level, specify the *level* argument as SOL_SOCKET. To set options at other levels, supply the appropriate *level* identifier for the protocol controlling the option. For example, to indicate that an option is interpreted by the TCP (Transport Control Protocol), set *level* to IPPROTO_TCP as defined in the *<netinet/in.h>* header.

The *option_name* argument specifies a single option to set. It can be one of the socket-level options defined in *<sys/socket.h>* and described in [Section 2.10.16](#) (on page 522). If *setsockopt()* is called with *option_name* equal to SO_ACCEPTCONN, SO_ERROR, or SO_TYPE, the behavior is unspecified.

RETURN VALUE

Upon successful completion, *setsockopt()* shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setsockopt()* function shall fail if:

[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
[EDOM]	The send and receive timeout values are too big to fit into the timeout fields in the socket structure.
[EINVAL]	The specified option is invalid at the specified socket level or the socket has been shut down.
[EISCONN]	The socket is already connected, and a specified option cannot be set while the socket is connected.
[ENOPROTOOPT]	The option is not supported by the protocol.
[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
- The *setsockopt()* function may fail if:
- | | |
|-----------|--|
| [ENOMEM] | There was insufficient memory available for the operation to complete. |
| [ENOBUFS] | Insufficient resources are available in the system to complete the call. |

setsockopt()

System Interfaces

60113 **EXAMPLES**

60114 None.

60115 **APPLICATION USAGE**

60116 The *setsockopt()* function provides an application program with the means to control socket
 60117 behavior. An application program can use *setsockopt()* to allocate buffer space, control timeouts,
 60118 or permit socket data broadcasts. The `<sys/socket.h>` header defines the socket-level options
 60119 available to *setsockopt()*.

60120 Options may exist at multiple protocol levels. The `SO_` options are always present at the
 60121 uppermost socket level.

60122 **RATIONALE**

60123 None.

60124 **FUTURE DIRECTIONS**

60125 None.

60126 **SEE ALSO**60127 [Section 2.10](#) (on page 517), *bind()*, *endprotoent()*, *getsockopt()*, *socket()*.60128 XBD `<netinet/in.h>`, `<sys/socket.h>`60129 **CHANGE HISTORY**

60130 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

60131 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/125 is applied, updating the `SO_LINGER`
 60132 option in the DESCRIPTION to refer to the calling thread rather than the process.

60133 **Issue 7**

60134 Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options
 60135 that is now in [Section 2.10.16](#) (on page 522).

*System Interfaces***setstate()**60136 **NAME**

60137 setstate — switch pseudo-random number generator state arrays

60138 **SYNOPSIS**

```
60139 XSI       #include <stdlib.h>  
60140       char *setstate(char *state);
```

60141 **DESCRIPTION**60142 Refer to *initstate()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

setuid()*System Interfaces*60143 **NAME**

60144 setuid — set user ID

60145 **SYNOPSIS**

60146 #include <unistd.h>

60147 int setuid(uid_t uid);

60148 **DESCRIPTION**60149 If the process has appropriate privileges, *setuid()* shall set the real user ID, effective user ID, and
60150 the saved set-user-ID of the calling process to *uid*.60151 If the process does not have appropriate privileges, but *uid* is equal to the real user ID or the
60152 saved set-user-ID, *setuid()* shall set the effective user ID to *uid*; the real user ID and saved set-
60153 user-ID shall remain unchanged.60154 The *setuid()* function shall not affect the supplementary group list in any way.60155 **RETURN VALUE**60156 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
60157 indicate the error.60158 **ERRORS**60159 The *setuid()* function shall fail, return -1, and set *errno* to the corresponding value if one or more
60160 of the following are true:60161 [EINVAL] The value of the *uid* argument is invalid and not supported by the
60162 implementation.60163 [EPERM] The process does not have appropriate privileges and *uid* does not match the
60164 real user ID or the saved set-user-ID.60165 **EXAMPLES**

60166 None.

60167 **APPLICATION USAGE**

60168 None.

60169 **RATIONALE**60170 The various behaviors of the *setuid()* and *setgid()* functions when called by non-privileged
60171 processes reflect the behavior of different historical implementations. For portability, it
60172 is recommended that new non-privileged applications use the *seteuid()* and *setegid()* functions
60173 instead.60174 The saved set-user-ID capability allows a program to regain the effective user ID established at
60175 the last *exec* call. Similarly, the saved set-group-ID capability allows a program to regain the
60176 effective group ID established at the last *exec* call. These capabilities are derived from System V.
60177 Without them, a program might have to run as superuser in order to perform the same
60178 functions, because superuser can write on the user's files. This is a problem because such a
60179 program can write on any user's files, and so must be carefully written to emulate the
60180 permissions of the calling process properly. In System V, these capabilities have traditionally
60181 been implemented only via the *setuid()* and *setgid()* functions for non-privileged processes. The
60182 fact that the behavior of those functions was different for privileged processes made them
60183 difficult to use. The POSIX.1-1990 standard defined the *setuid()* function to behave differently
60184 for privileged and unprivileged users. When the caller had appropriate privileges, the function
60185 set the real user ID, effective user ID, and saved set-user ID of the calling process on
60186 implementations that supported it. When the caller did not have appropriate privileges, the
60187 function set only the effective user ID, subject to permission checks. The former use is generally
60188 needed for utilities like *login* and *su*, which are not conforming applications and thus outside the

scope of POSIX.1-2008. These utilities wish to change the user ID irrevocably to a new value, generally that of an unprivileged user. The latter use is needed for conforming applications that are installed with the set-user-ID bit and need to perform operations using the real user ID.

POSIX.1-2008 augments the latter functionality with a mandatory feature named `_POSIX_SAVED_IDS`. This feature permits a set-user-ID application to switch its effective user ID back and forth between the values of its *exec*-time real user ID and effective user ID. Unfortunately, the POSIX.1-1990 standard did not permit a conforming application using this feature to work properly when it happened to be executed with (implementation-defined) appropriate privileges. Furthermore, the application did not even have a means to tell whether it had this privilege. Since the saved set-user-ID feature is quite desirable for applications, as evidenced by the fact that NIST required it in FIPS 151-2, it has been mandated by POSIX.1-2008. However, there are implementors who have been reluctant to support it given the limitation described above.

The 4.3BSD system handles the problem by supporting separate functions: *setuid()* (which always sets both the real and effective user IDs, like *setuid()* in POSIX.1-2008 for privileged users), and *seteuid()* (which always sets just the effective user ID, like *setuid()* in POSIX.1-2008 for non-privileged users). This separation of functionality into distinct functions seems desirable. 4.3BSD does not support the saved set-user-ID feature. It supports similar functionality of switching the effective user ID back and forth via *setreuid()*, which permits reversing the real and effective user IDs. This model seems less desirable than the saved set-user-ID because the real user ID changes as a side-effect. The current 4.4BSD includes saved effective IDs and uses them for *seteuid()* and *setegid()* as described above. The *setreuid()* and *setregid()* functions will be deprecated or removed.

The solution here is:

- Require that all implementations support the functionality of the saved set-user-ID, which is set by the *exec* functions and by privileged calls to *setuid()*.
- Add the *seteuid()* and *setegid()* functions as portable alternatives to *setuid()* and *setgid()* for non-privileged and privileged processes.

Historical systems have provided two mechanisms for a set-user-ID process to change its effective user ID to be the same as its real user ID in such a way that it could return to the original effective user ID: the use of the *setuid()* function in the presence of a saved set-user-ID, or the use of the BSD *setreuid()* function, which was able to swap the real and effective user IDs. The changes included in POSIX.1-2008 provide a new mechanism using *seteuid()* in conjunction with a saved set-user-ID. Thus, all implementations with the new *seteuid()* mechanism will have a saved set-user-ID for each process, and most of the behavior controlled by `_POSIX_SAVED_IDS` has been changed to agree with the case where the option was defined. The *kill()* function is an exception. Implementors of the new *seteuid()* mechanism will generally be required to maintain compatibility with the older mechanisms previously supported by their systems. However, compatibility with this use of *setreuid()* and with the `_POSIX_SAVED_IDS` behavior of *kill()* is unfortunately complicated. If an implementation with a saved set-user-ID allows a process to use *setreuid()* to swap its real and effective user IDs, but were to leave the saved set-user-ID unmodified, the process would then have an effective user ID equal to the original real user ID, and both real and saved set-user-ID would be equal to the original effective user ID. In that state, the real user would be unable to kill the process, even though the effective user ID of the process matches that of the real user, if the *kill()* behavior of `_POSIX_SAVED_IDS` was used. This is obviously not acceptable. The alternative choice, which is used in at least one implementation, is to change the saved set-user-ID to the effective user ID during most calls to *setreuid()*. The standard developers considered that alternative to be less correct than the retention of the old behavior of *kill()* in such systems. Current conforming applications shall

setuid()

System Interfaces

60238 accommodate either behavior from *kill()*, and there appears to be no strong reason for *kill()* to
 60239 check the saved set-user-ID rather than the effective user ID.

60240 **FUTURE DIRECTIONS**

60241 None.

60242 **SEE ALSO**

60243 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*

60244 XBD *<sys/types.h>*, *<unistd.h>*

60245 **CHANGE HISTORY**

60246 First released in Issue 1. Derived from Issue 1 of the SVID.

60247 **Issue 6**

60248 In the SYNOPSIS, the optional include of the *<sys/types.h>* header is removed.

60249 The following new requirements on POSIX implementations derive from alignment with the
 60250 Single UNIX Specification:

- 60251 • The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was
 60252 required for conforming implementations of previous POSIX specifications, it was not
 60253 required for UNIX applications.
- 60254 • The functionality associated with *_POSIX_SAVED_IDS* is now mandatory. This is a FIPS
 60255 requirement.

60256 The following changes were made to align with the IEEE P1003.1a draft standard:

- 60257 • The effects of *setuid()* in processes without appropriate privileges are changed.
- 60258 • A requirement that the supplementary group list is not affected is added.

*System Interfaces***setutxent()**60259 **NAME**

60260 setutxent — reset the user accounting database to the first entry

60261 **SYNOPSIS**

```
60262 XSI      #include <utmpx.h>  
60263          void setutxent(void);
```

60264 **DESCRIPTION**60265 Refer to *endutxent()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

setvbuf()*System Interfaces*60266 **NAME**

60267 setvbuf — assign buffering to a stream

60268 **SYNOPSIS**

60269 #include <stdio.h>

```
60270       int setvbuf(FILE *restrict stream, char *restrict buf, int type,
60271                   size_t size);
```

60272 **DESCRIPTION**

60273 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 60274 conflict between the requirements described here and the ISO C standard is unintentional. This
 60275 volume of POSIX.1-2008 defers to the ISO C standard.

60276 The *setvbuf()* function may be used after the stream pointed to by *stream* is associated with an
 60277 open file but before any other operation (other than an unsuccessful call to *setvbuf()*) is
 60278 performed on the stream. The argument *type* determines how *stream* shall be buffered, as
 60279 follows:

- 60280 • {_IOFBF} shall cause input/output to be fully buffered.
- 60281 • {_IOLBF} shall cause input/output to be line buffered.
- 60282 • {_IONBF} shall cause input/output to be unbuffered.

60283 If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by
 60284 *setvbuf()* and the argument *size* specifies the size of the array; otherwise, *size* may determine the
 60285 size of a buffer allocated by the *setvbuf()* function. The contents of the array at any time are
 60286 unspecified.

60287 For information about streams, see [Section 2.5](#) (on page 490).

60288 **RETURN VALUE**

60289 Upon successful completion, *setvbuf()* shall return 0. Otherwise, it shall return a non-zero value
 60290 CX if an invalid value is given for *type* or if the request cannot be honored, and may set *errno* to
 60291 indicate the error.

60292 **ERRORS**

60293 The *setvbuf()* function may fail if:

60294 CX [EBADF] The file descriptor underlying *stream* is not valid.

60295 **EXAMPLES**

60296 None.

60297 **APPLICATION USAGE**

60298 A common source of error is allocating buffer space as an “automatic” variable in a code block,
 60299 and then failing to close the stream in the same block.

60300 With *setvbuf()*, allocating a buffer of *size* bytes does not necessarily imply that all of *size* bytes are
 60301 used for the buffer area.

60302 Applications should note that many implementations only provide line buffering on input from
 60303 terminal devices.

60304 **RATIONALE**

60305 None.

60306 **FUTURE DIRECTIONS**

60307 None.

60308 **SEE ALSO**60309 [Section 2.5](#) (on page 490), *fopen()*, *setbuf()*60310 XBD [<stdio.h>](#)60311 **CHANGE HISTORY**

60312 First released in Issue 1. Derived from Issue 1 of the SVID.

60313 **Issue 6**

60314 Extensions beyond the ISO C standard are marked.

60315 The *setvbuf()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

shm_open()*System Interfaces*60316 **NAME**60317 shm_open — open a shared memory object (**REALTIME**)60318 **SYNOPSIS**

```
60319 SHM    #include <sys/mman.h>
60320
        int shm_open(const char *name, int oflag, mode_t mode);
```

60321 **DESCRIPTION**

60322 The *shm_open()* function shall establish a connection between a shared memory object and a file
 60323 descriptor. It shall create an open file description that refers to the shared memory object and a
 60324 file descriptor that refers to that open file description. The file descriptor is used by other
 60325 functions to refer to that shared memory object. The *name* argument points to a string naming a
 60326 shared memory object. It is unspecified whether the name appears in the file system and is
 60327 visible to other functions that take pathnames as arguments. The *name* argument conforms to the
 60328 construction rules for a pathname, except that the interpretation of <slash> characters other than
 60329 the leading <slash> character in *name* is implementation-defined, and that the length limits for
 60330 the *name* argument are implementation-defined and need not be the same as the pathname limits
 60331 {PATH_MAX} and {NAME_MAX}. If *name* begins with the <slash> character, then processes
 60332 calling *shm_open()* with the same value of *name* refer to the same shared memory object, as long
 60333 as that name has not been removed. If *name* does not begin with the <slash> character, the effect
 60334 is implementation-defined.

60335 If successful, *shm_open()* shall return a file descriptor for the shared memory object that is the
 60336 lowest numbered file descriptor not currently open for that process. The open file description is
 60337 new, and therefore the file descriptor does not share it with any other processes. It is unspecified
 60338 whether the file offset is set. The FD_CLOEXEC file descriptor flag associated with the new file
 60339 descriptor is set.

60340 The file status flags and file access modes of the open file description are according to the value
 60341 of *oflag*. The *oflag* argument is the bitwise-inclusive OR of the following flags defined in the
 60342 <fcntl.h> header. Applications specify exactly one of the first two values (access modes) below
 60343 in the value of *oflag*:

60344 O_RDONLY Open for read access only.

60345 O_RDWR Open for read or write access.

60346 Any combination of the remaining flags may be specified in the value of *oflag*:

60347 O_CREAT If the shared memory object exists, this flag has no effect, except as noted
 60348 under O_EXCL below. Otherwise, the shared memory object is created. The
 60349 user ID of the shared memory object shall be set to the effective user ID of the
 60350 process. The group ID of the shared memory object shall be set to the effective
 60351 group ID of the process; however, if the *name* argument is visible in the file
 60352 system, the group ID may be set to the group ID of the containing directory.
 60353 The permission bits of the shared memory object shall be set to the value of
 60354 the *mode* argument except those set in the file mode creation mask of the
 60355 process. When bits in *mode* other than the file permission bits are set, the effect
 60356 is unspecified. The *mode* argument does not affect whether the shared memory
 60357 object is opened for reading, for writing, or for both. The shared memory
 60358 object has a size of zero.

60359 O_EXCL If O_EXCL and O_CREAT are set, *shm_open()* fails if the shared memory
 60360 object exists. The check for the existence of the shared memory object and the
 60361 creation of the object if it does not exist is atomic with respect to other

System Interfaces

shm_open()

processes executing *shm_open()* naming the same shared memory object with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is undefined.

O_TRUNC If the shared memory object exists, and it is successfully opened O_RDWR, the object shall be truncated to zero length and the mode and owner shall be unchanged by this function call. The result of using O_TRUNC with O_RDONLY is undefined.

When a shared memory object is created, the state of the shared memory object, including all data associated with the shared memory object, persists until the shared memory object is unlinked and all other references are gone. It is unspecified whether the name and shared memory object state remain valid after a system reboot.

RETURN VALUE

Upon successful completion, the *shm_open()* function shall return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, it shall return -1 and set *errno* to indicate the error.

ERRORS

The *shm_open()* function shall fail if:

[EACCES] The shared memory object exists and the permissions specified by *oflag* are denied, or the shared memory object does not exist and permission to create the shared memory object is denied, or O_TRUNC is specified and write permission is denied.

[EEXIST] O_CREAT and O_EXCL are set and the named shared memory object already exists.

[EINTR] The *shm_open()* operation was interrupted by a signal.

[EINVAL] The *shm_open()* operation is not supported for the given name.

[EMFILE] All file descriptors available to the process are currently open.

[ENFILE] Too many shared memory objects are currently open in the system.

[ENOENT] O_CREAT is not set and the named shared memory object does not exist.

[ENOSPC] There is insufficient space for the creation of the new shared memory object.

The *shm_open()* function may fail if:

[ENAMETOOLONG]

The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option or longer than {_XOPEN_NAME_MAX} on XSI systems.

shm_open()*System Interfaces*60398 **EXAMPLES**60399 **Creating and Mapping a Shared Memory Object**

60400 The following code segment demonstrates the use of *shm_open()* to create a shared memory
 60401 object which is then sized using *ftruncate()* before being mapped into the process address space
 60402 using *mmap()*:

```

60403 #include <unistd.h>
60404 #include <sys/mman.h>
60405 ...
60406 #define MAX_LEN 10000
60407 struct region {          /* Defines "structure" of shared memory */
60408     int len;
60409     char buf[MAX_LEN];
60410 };
60411 struct region *rptra;
60412 int fd;
60413
60414 /* Create shared memory object and set its size */
60415 fd = shm_open("/myregion", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
60416 if (fd == -1)
60417     /* Handle error */;
60418
60419 if (ftruncate(fd, sizeof(struct region)) == -1)
60420     /* Handle error */;
60421
60422 /* Map shared memory object */
60423 rptra = mmap(NULL, sizeof(struct region),
60424             PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
60425 if (rptra == MAP_FAILED)
60426     /* Handle error */;
60427
60428 /* Now we can refer to mapped region using fields of rptra;
60429    for example, rptra->len */
60430 ...
  
```

60427 **APPLICATION USAGE**

60428 None.

60429 **RATIONALE**

60430 When the Memory Mapped Files option is supported, the normal *open()* call is used to obtain a
 60431 descriptor to a file to be mapped according to existing practice with *mmap()*. When the Shared
 60432 Memory Objects option is supported, the *shm_open()* function shall obtain a descriptor to the
 60433 shared memory object to be mapped.

60434 There is ample precedent for having a file descriptor represent several types of objects. In the
 60435 POSIX.1-1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory.
 60436 Many implementations simply have an operations vector, which is indexed by the file descriptor
 60437 type and does very different operations. Note that in some cases the file descriptor passed to
 60438 generic operations on file descriptors is returned by *open()* or *creat()* and in some cases returned
 60439 by alternate functions, such as *pipe()*. The latter technique is used by *shm_open()*.

60440 Note that such shared memory objects can actually be implemented as mapped files. In both
 60441 cases, the size can be set after the open using *ftruncate()*. The *shm_open()* function itself does not

create a shared object of a specified size because this would duplicate an extant function that set the size of an object referenced by a file descriptor.

On implementations where memory objects are implemented using the existing file system, the *shm_open()* function may be implemented using a macro that invokes *open()*, and the *shm_unlink()* function may be implemented using a macro that invokes *unlink()*.

For implementations without a permanent file system, the definition of the name of the memory objects is allowed not to survive a system reboot. Note that this allows systems with a permanent file system to implement memory objects as data structures internal to the implementation as well.

On implementations that choose to implement memory objects using memory directly, a *shm_open()* followed by an *ftruncate()* and *close()* can be used to preallocate a shared memory area and to set the size of that preallocation. This may be necessary for systems without virtual memory hardware support in order to ensure that the memory is contiguous.

The set of valid open flags to *shm_open()* was restricted to *O_RDONLY*, *O_RDWR*, *O_CREAT*, and *O_TRUNC* because these could be easily implemented on most memory mapping systems. This volume of POSIX.1-2008 is silent on the results if the implementation cannot supply the requested file access because of implementation-defined reasons, including hardware ones.

The error conditions [EACCES] and [ENOTSUP] are provided to inform the application that the implementation cannot complete a request.

[EACCES] indicates for implementation-defined reasons, probably hardware-related, that the implementation cannot comply with a requested mode because it conflicts with another requested mode. An example might be that an application desires to open a memory object two times, mapping different areas with different access modes. If the implementation cannot map a single area into a process space in two places, which would be required if different access modes were required for the two areas, then the implementation may inform the application at the time of the second open.

[ENOTSUP] indicates for implementation-defined reasons, probably hardware-related, that the implementation cannot comply with a requested mode at all. An example would be that the hardware of the implementation cannot support write-only shared memory areas.

On all implementations, it may be desirable to restrict the location of the memory objects to specific file systems for performance (such as a RAM disk) or implementation-defined reasons (shared memory supported directly only on certain file systems). The *shm_open()* function may be used to enforce these restrictions. There are a number of methods available to the application to determine an appropriate name of the file or the location of an appropriate directory. One way is from the environment via *getenv()*. Another would be from a configuration file.

This volume of POSIX.1-2008 specifies that memory objects have initial contents of zero when created. This is consistent with current behavior for both files and newly allocated memory. For those implementations that use physical memory, it would be possible that such implementations could simply use available memory and give it to the process uninitialized. This, however, is not consistent with standard behavior for the uninitialized data area, the stack, and of course, files. Finally, it is highly desirable to set the allocated memory to zero for security reasons. Thus, initializing memory objects to zero is required.

FUTURE DIRECTIONS

A future version might require the *shm_open()* and *shm_unlink()* functions to have semantics similar to normal file system operations.

shm_open()

System Interfaces

60487 **SEE ALSO**60488 *close()*, *dup()*, *exec*, *fcntl()*, *mmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm_unlink()*, *umask()*

60489 XBD <fcntl.h>, <sys/mman.h>

60490 **CHANGE HISTORY**

60491 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

60492 **Issue 6**60493 The *shm_open()* function is marked as part of the Shared Memory Objects option.60494 The [ENOSYS] error condition has been removed as stubs need not be provided if an
60495 implementation does not support the Shared Memory Objects option.60496 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/126 is applied, adding the example to the
60497 EXAMPLES section.60498 **Issue 7**60499 Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and
60500 changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

60501 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

60502 SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the
60503 user ID and group ID of the shared memory object.

NAME

shm_unlink — remove a shared memory object (**REALTIME**)

SYNOPSIS

```
#include <sys/mman.h>

int shm_unlink(const char *name);
```

DESCRIPTION

The *shm_unlink()* function shall remove the name of the shared memory object named by the string pointed to by *name*.

If one or more references to the shared memory object exist when the object is unlinked, the name shall be removed before *shm_unlink()* returns, but the removal of the memory object contents shall be postponed until all open and map references to the shared memory object have been removed.

Even if the object continues to exist after the last *shm_unlink()*, reuse of the name shall subsequently cause *shm_open()* to behave as if no shared memory object of this name exists (that is, *shm_open()* will fail if **O_CREAT** is not set, or will create a new shared memory object if **O_CREAT** is set).

RETURN VALUE

Upon successful completion, a value of zero shall be returned. Otherwise, a value of **-1** shall be returned and *errno* set to indicate the error. If **-1** is returned, the named shared memory object shall not be changed by this function call.

ERRORS

The *shm_unlink()* function shall fail if:

[EACCES] Permission is denied to unlink the named shared memory object.

[ENOENT] The named shared memory object does not exist.

The *shm_unlink()* function may fail if:

[ENAMETOOLONG]

The length of the *name* argument exceeds **{_POSIX_PATH_MAX}** on systems that do not support the XSI option or exceeds **{_XOPEN_PATH_MAX}** on XSI systems, or has a pathname component that is longer than **{_POSIX_NAME_MAX}** on systems that do not support the XSI option or longer than **{_XOPEN_NAME_MAX}** on XSI systems. A call to *shm_unlink()* with a *name* argument that contains the same shared memory object name as was previously used in a successful *shm_open()* call shall not give an [ENAMETOOLONG] error.

EXAMPLES

None.

APPLICATION USAGE

Names of memory objects that were allocated with *open()* are deleted with *unlink()* in the usual fashion. Names of memory objects that were allocated with *shm_open()* are deleted with *shm_unlink()*. Note that the actual memory object is not destroyed until the last close and unmap on it have occurred if it was already in use.

shm_unlink()

System Interfaces

60545 **RATIONALE**

60546 None.

60547 **FUTURE DIRECTIONS**

60548 A future version might require the *shm_open()* and *shm_unlink()* functions to have semantics
60549 similar to normal file system operations.

60550 **SEE ALSO**60551 *close()*, *mmap()*, *munmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm_open()*

60552 XBD <sys/mman.h>

60553 **CHANGE HISTORY**

60554 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

60555 **Issue 6**60556 The *shm_unlink()* function is marked as part of the Shared Memory Objects option.

60557 In the DESCRIPTION, text is added to clarify that reusing the same name after a *shm_unlink()*
60558 will not attach to the old shared memory object.

60559 The [ENOSYS] error condition has been removed as stubs need not be provided if an
60560 implementation does not support the Shared Memory Objects option.

60561 **Issue 7**

60562 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a
60563 “shall fail” to a “may fail” error.

60564 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

NAME

shmat — XSI shared memory attach operation

SYNOPSIS

```
#include <sys/shm.h>

void *shmat(int shmid, const void *shmaddr, int shmflg);
```

DESCRIPTION

The *shmat()* function operates on XSI shared memory (see XBD [Section 3.340](#), on page 88). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in [Section 2.8](#) (on page 497).

The *shmat()* function attaches the shared memory segment associated with the shared memory identifier specified by *shmid* to the address space of the calling process. The segment is attached at the address specified by one of the following criteria:

- If *shmaddr* is a null pointer, the segment is attached at the first available address as selected by the system.
- If *shmaddr* is not a null pointer and (*shmflg* & SHM_RND) is non-zero, the segment is attached at the address given by (*shmaddr* - ((*uintptr_t*)*shmaddr* % SHMLBA)). The character '%' is the C-language remainder operator.
- If *shmaddr* is not a null pointer and (*shmflg* & SHM_RND) is 0, the segment is attached at the address given by *shmaddr*.
- The segment is attached for reading if (*shmflg* & SHM_RDONLY) is non-zero and the calling process has read permission; otherwise, if it is 0 and the calling process has read and write permission, the segment is attached for reading and writing.

RETURN VALUE

Upon successful completion, *shmat()* shall increment the value of *shm_nattch* in the data structure associated with the shared memory ID of the attached shared memory segment and return the segment's start address.

Otherwise, the shared memory segment shall not be attached, *shmat()* shall return -1, and *errno* shall be set to indicate the error.

ERRORS

The *shmat()* function shall fail if:

[EACCES]	Operation permission is denied to the calling process; see Section 2.7 (on page 496).
[EINVAL]	The value of <i>shmid</i> is not a valid shared memory identifier, the <i>shmaddr</i> is not a null pointer, and the value of (<i>shmaddr</i> - ((<i>uintptr_t</i>) <i>shmaddr</i> % SHMLBA)) is an illegal address for attaching shared memory; or the <i>shmaddr</i> is not a null pointer, (<i>shmflg</i> & SHM_RND) is 0, and the value of <i>shmaddr</i> is an illegal address for attaching shared memory.
[EMFILE]	The number of shared memory segments attached to the calling process would exceed the system-imposed limit.
[ENOMEM]	The available data space is not large enough to accommodate the shared memory segment.

shmat()*System Interfaces*60606 **EXAMPLES**

60607 None.

60608 **APPLICATION USAGE**

60609 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
 60610 Application developers who need to use IPC should design their applications so that modules
 60611 using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to use the
 60612 alternative interfaces.

60613 **RATIONALE**

60614 None.

60615 **FUTURE DIRECTIONS**

60616 None.

60617 **SEE ALSO**

60618 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), *exec*, *exit()*, *fork()*, *shmctl()*, *shmdt()*,
 60619 *shmget()*, *shm_open()*, *shm_unlink()*

60620 XBD [Section 3.340](#) (on page 88), [<sys/shm.h>](#)60621 **CHANGE HISTORY**

60622 First released in Issue 2. Derived from Issue 2 of the SVID.

60623 **Issue 5**

60624 Moved from SHARED MEMORY to BASE.

60625 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
 60626 DIRECTIONS to a new APPLICATION USAGE section.

60627 **Issue 6**

60628 The Open Group Corrigendum U021/13 is applied.

NAME

shmctl — XSI shared memory control operations

SYNOPSIS

```

#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shm_id *buf);

```

DESCRIPTION

The *shmctl()* function operates on XSI shared memory (see XBD [Section 3.340](#), on page 88). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in [Section 2.8](#) (on page 497).

The *shmctl()* function provides a variety of shared memory control operations as specified by *cmd*. The following values for *cmd* are available:

IPC_STAT Place the current value of each member of the **shm_id** data structure associated with *shmid* into the structure pointed to by *buf*. The contents of the structure are defined in **<sys/shm.h>**.

IPC_SET Set the value of the following members of the **shm_id** data structure associated with *shmid* to the corresponding value found in the structure pointed to by *buf*:

```

shm_perm.uid
shm_perm.gid
shm_perm.mode    Low-order nine bits.

```

IPC_SET can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *shm_perm.cuid* or *shm_perm.uid* in the **shm_id** data structure associated with *shmid*.

IPC_RMID Remove the shared memory identifier specified by *shmid* from the system and destroy the shared memory segment and **shm_id** data structure associated with it. **IPC_RMID** can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *shm_perm.cuid* or *shm_perm.uid* in the **shm_id** data structure associated with *shmid*.

RETURN VALUE

Upon successful completion, *shmctl()* shall return 0; otherwise, it shall return -1 and set *errno* to indicate the error.

ERRORS

The *shmctl()* function shall fail if:

[EACCES] The argument *cmd* is equal to **IPC_STAT** and the calling process does not have read permission; see [Section 2.7](#) (on page 496).

[EINVAL] The value of *shmid* is not a valid shared memory identifier, or the value of *cmd* is not a valid command.

[EPERM] The argument *cmd* is equal to **IPC_RMID** or **IPC_SET** and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of *shm_perm.cuid* or *shm_perm.uid* in the data structure associated with *shmid*.

shmctl()*System Interfaces*

60672 The *shmctl()* function may fail if:
 60673 [EOVERFLOW] The *cmd* argument is IPC_STAT and the *gid* or *uid* value is too large to be
 60674 stored in the structure pointed to by the *buf* argument.

60675 **EXAMPLES**

60676 None.

60677 **APPLICATION USAGE**

60678 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
 60679 Application developers who need to use IPC should design their applications so that modules
 60680 using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to use the
 60681 alternative interfaces.

60682 **RATIONALE**

60683 None.

60684 **FUTURE DIRECTIONS**

60685 None.

60686 **SEE ALSO**

60687 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), *shmat()*, *shmdt()*, *shmget()*, *shm_open()*,
 60688 *shm_unlink()*

60689 XBD [Section 3.340](#) (on page 88), [<sys/shm.h>](#)60690 **CHANGE HISTORY**

60691 First released in Issue 2. Derived from Issue 2 of the SVID.

60692 **Issue 5**

60693 Moved from SHARED MEMORY to BASE.

60694 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
 60695 DIRECTIONS to a new APPLICATION USAGE section.

NAME

shmdt — XSI shared memory detach operation

SYNOPSIS

```
#include <sys/shm.h>

int shmdt(const void *shmaddr);
```

DESCRIPTION

The *shmdt()* function operates on XSI shared memory (see XBD Section 3.340, on page 88). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 497).

The *shmdt()* function detaches the shared memory segment located at the address specified by *shmaddr* from the address space of the calling process.

RETURN VALUE

Upon successful completion, *shmdt()* shall decrement the value of *shm_nattch* in the data structure associated with the shared memory ID of the attached shared memory segment and return 0.

Otherwise, the shared memory segment shall not be detached, *shmdt()* shall return -1, and *errno* shall be set to indicate the error.

ERRORS

The *shmdt()* function shall fail if:

[EINVAL]	The value of <i>shmaddr</i> is not the data segment start address of a shared memory segment.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 496) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 496), Section 2.8 (on page 497), *exec*, *exit()*, *fork()*, *shmat()*, *shmctl()*, *shmget()*, *shm_open()*, *shm_unlink()*

XBD Section 3.340 (on page 88), *<sys/shm.h>*

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

shmdt()*System Interfaces*60734 **Issue 5**

60735 Moved from SHARED MEMORY to BASE.

60736 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
60737 DIRECTIONS to a new APPLICATION USAGE section.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

shmget — get an XSI shared memory segment

SYNOPSIS

```
XSI #include <sys/shm.h>

int shmget(key_t key, size_t size, int shmflg);
```

DESCRIPTION

The *shmget()* function operates on XSI shared memory (see XBD [Section 3.340](#), on page 88). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in [Section 2.8](#) (on page 497).

The *shmget()* function shall return the shared memory identifier associated with *key*.

A shared memory identifier, associated data structure, and shared memory segment of at least *size* bytes (see [<sys/shm.h>](#)) are created for *key* if one of the following is true:

- The argument *key* is equal to `IPC_PRIVATE`.
- The argument *key* does not already have a shared memory identifier associated with it and (*shmflg* & `IPC_CREAT`) is non-zero.

Upon creation, the data structure associated with the new shared memory identifier shall be initialized as follows:

- The values of *shm_perm.cuid*, *shm_perm.uid*, *shm_perm.cgid*, and *shm_perm.gid* are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order nine bits of *shm_perm.mode* are set equal to the low-order nine bits of *shmflg*.
- The value of *shm_segsz* is set equal to the value of *size*.
- The values of *shm_lpid*, *shm_nattch*, *shm_atime*, and *shm_dtime* are set equal to 0.
- The value of *shm_ctime* is set equal to the current time.

When the shared memory segment is created, it shall be initialized with all zero values.

RETURN VALUE

Upon successful completion, *shmget()* shall return a non-negative integer, namely a shared memory identifier; otherwise, it shall return -1 and set *errno* to indicate the error.

ERRORS

The *shmget()* function shall fail if:

[EACCES]	A shared memory identifier exists for <i>key</i> but operation permission as specified by the low-order nine bits of <i>shmflg</i> would not be granted; see Section 2.7 (on page 496).
[EEXIST]	A shared memory identifier exists for the argument <i>key</i> but (<i>shmflg</i> & <code>IPC_CREAT</code>) && (<i>shmflg</i> & <code>IPC_EXCL</code>) is non-zero.
[EINVAL]	A shared memory segment is to be created and the value of <i>size</i> is less than the system-imposed minimum or greater than the system-imposed maximum.
[EINVAL]	No shared memory segment is to be created and a shared memory segment exists for <i>key</i> but the size of the segment associated with it is less than <i>size</i> and <i>size</i> is not 0.

shmget()*System Interfaces*

60777	[ENOENT]	A shared memory identifier does not exist for the argument <i>key</i> and (<i>shmflg</i> & IPC_CREAT) is 0.
60778		
60779	[ENOMEM]	A shared memory identifier and associated shared memory segment shall be created, but the amount of available physical memory is not sufficient to fill the request.
60780		
60781		
60782	[ENOSPC]	A shared memory identifier is to be created, but the system-imposed limit on the maximum number of allowed shared memory identifiers system-wide would be exceeded.
60783		
60784		

EXAMPLES

60785
60786 None.

APPLICATION USAGE

60787 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
60788 Application developers who need to use IPC should design their applications so that modules
60789 using the IPC routines described in [Section 2.7](#) (on page 496) can be easily modified to use the
60790 alternative interfaces.
60791

RATIONALE

60792
60793 None.

FUTURE DIRECTIONS

60794
60795 None.

SEE ALSO

60796 [Section 2.7](#) (on page 496), [Section 2.8](#) (on page 497), [shmat\(\)](#), [shmctl\(\)](#), [shmdt\(\)](#), [shm_open\(\)](#),
60797 [shm_unlink\(\)](#)
60798
60799 XBD [Section 3.340](#) (on page 88), [<sys/shm.h>](#)

CHANGE HISTORY

60800 First released in Issue 2. Derived from Issue 2 of the SVID.
60801

Issue 5

60802 Moved from SHARED MEMORY to BASE.
60803

60804 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
60805 DIRECTIONS to a new APPLICATION USAGE section.

NAME

shutdown — shut down socket send and receive operations

SYNOPSIS

```
#include <sys/socket.h>

int shutdown(int socket, int how);
```

DESCRIPTION

The *shutdown()* function shall cause all or part of a full-duplex connection on the socket associated with the file descriptor *socket* to be shut down.

The *shutdown()* function takes the following arguments:

<i>socket</i>	Specifies the file descriptor of the socket.
<i>how</i>	Specifies the type of shutdown. The values are as follows:
SHUT_RD	Disables further receive operations.
SHUT_WR	Disables further send operations.
SHUT_RDWR	Disables further send and receive operations.

The *shutdown()* function disables subsequent send and/or receive operations on a socket, depending on the value of the *how* argument.

RETURN VALUE

Upon successful completion, *shutdown()* shall return 0; otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *shutdown()* function shall fail if:

[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
[EINVAL]	The <i>how</i> argument is invalid.
[ENOTCONN]	The socket is not connected.
[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.

The *shutdown()* function may fail if:

[ENOBUFS]	Insufficient resources were available in the system to perform the operation.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getsockopt(), *pselect()*, *read()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendto()*, *setsockopt()*, *socket()*, *write()*

XBD [<sys/socket.h>](#)

shutdown()

System Interfaces

CHANGE HISTORY

60845
60846

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

sigaction()**NAME**

sigaction — examine and change a signal action

SYNOPSIS

```
#include <signal.h>

int sigaction(int sig, const struct sigaction *restrict act,
              struct sigaction *restrict oact);
```

DESCRIPTION

The *sigaction()* function allows the calling process to examine and/or specify the action to be associated with a specific signal. The argument *sig* specifies the signal; acceptable values are defined in **<signal.h>**.

The structure **sigaction**, used to describe an action to be taken, is defined in the **<signal.h>** header to include at least the following members:

Member Type	Member Name	Description
void(*) (int)	<i>sa_handler</i>	Pointer to a signal-catching function or one of the macros SIG_IGN or SIG_DFL.
sigset_t	<i>sa_mask</i>	Additional set of signals to be blocked during execution of signal-catching function.
int	<i>sa_flags</i>	Special flags to affect behavior of signal.
void(*) (int, siginfo_t *, void *)	<i>sa_sigaction</i>	Pointer to a signal-catching function.

The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a conforming application shall not use both simultaneously.

If the argument *act* is not a null pointer, it points to a structure specifying the action to be associated with the specified signal. If the argument *oact* is not a null pointer, the action previously associated with the signal is stored in the location pointed to by the argument *oact*. If the argument *act* is a null pointer, signal handling is unchanged; thus, the call can be used to enquire about the current handling of a given signal. The SIGKILL and SIGSTOP signals shall not be added to the signal mask using this mechanism; this restriction shall be enforced by the system without causing an error to be indicated.

If the SA_SIGINFO flag (see below) is cleared in the *sa_flags* field of the **sigaction** structure, the *sa_handler* field identifies the action to be associated with the specified signal. If the SA_SIGINFO flag is set in the *sa_flags* field, the *sa_sigaction* field specifies a signal-catching function.

The *sa_flags* field can be used to modify the behavior of the specified signal.

The following flags, defined in the **<signal.h>** header, can be set in *sa_flags*:

SA_NOCLDSTOP Do not generate SIGCHLD when children stop or stopped children continue.

If *sig* is SIGCHLD and the SA_NOCLDSTOP flag is not set in *sa_flags*, and the implementation supports the SIGCHLD signal, then a SIGCHLD signal shall be generated for the calling process whenever any of its child processes stop and a SIGCHLD signal may be generated for the calling process whenever any of its stopped child processes are continued. If *sig* is SIGCHLD and the SA_NOCLDSTOP flag is set in *sa_flags*, then the implementation shall not generate a SIGCHLD signal in this way.

sigaction()*System Interfaces*

60892	XSI	SA_ONSTACK	If set and an alternate signal stack has been declared with <i>sigaltstack()</i> , the signal shall be delivered to the calling process on that stack. Otherwise, the signal shall be delivered on the current stack.
60893			
60894			
60895		SA_RESETHAND	If set, the disposition of the signal shall be reset to SIG_DFL and the SA_SIGINFO flag shall be cleared on entry to the signal handler.
60896			
60897		Note:	SIGILL and SIGTRAP cannot be automatically reset when delivered; the system silently enforces this restriction.
60898			
60899			Otherwise, the disposition of the signal shall not be modified on entry to the signal handler.
60900			
60901			In addition, if this flag is set, <i>sigaction()</i> may behave as if the SA_NODEFER flag were also set.
60902			
60903		SA_RESTART	This flag affects the behavior of interruptible functions; that is, those specified to fail with <i>errno</i> set to [EINTR]. If set, and a function specified as interruptible is interrupted by this signal, the function shall restart and shall not fail with [EINTR] unless otherwise specified. If an interruptible function which uses a timeout is restarted, the duration of the timeout following the restart is set to an unspecified value that does not exceed the original timeout value. If the flag is not set, interruptible functions interrupted by this signal shall fail with <i>errno</i> set to [EINTR].
60904			
60905			
60906			
60907			
60908			
60909			
60910			
60911		SA_SIGINFO	If cleared and the signal is caught, the signal-catching function shall be entered as:
60912			
60913			<code>void func(int signo);</code>
60914			where <i>signo</i> is the only argument to the signal-catching function. In this case, the application shall use the <i>sa_handler</i> member to describe the signal-catching function and the application shall not modify the <i>sa_sigaction</i> member.
60915			
60916			
60917			
60918			If SA_SIGINFO is set and the signal is caught, the signal-catching function shall be entered as:
60919			
60920			<code>void func(int signo, siginfo_t *info, void *context);</code>
60921			where two additional arguments are passed to the signal-catching function. The second argument shall point to an object of type siginfo_t explaining the reason why the signal was generated; the third argument can be cast to a pointer to an object of type ucontext_t to refer to the receiving thread's context that was interrupted when the signal was delivered. In this case, the application shall use the <i>sa_sigaction</i> member to describe the signal-catching function and the application shall not modify the <i>sa_handler</i> member.
60922			
60923			
60924			
60925			
60926			
60927			
60928			
60929			The <i>si_signo</i> member contains the system-generated signal number.
60930	XSI		The <i>si_errno</i> member may contain implementation-defined additional error information; if non-zero, it contains an error number identifying the condition that caused the signal to be generated.
60931			
60932			
60933			The <i>si_code</i> member contains a code identifying the cause of the signal, as described in Section 2.4.3 (on page 486).
60934			

SA_NOCLDWAIT If set, and *sig* equals SIGCHLD, child processes of the calling processes shall not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited-for children that were transformed into zombie processes, it shall block until all of its children terminate, and *wait()*, *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD]. Otherwise, terminating child processes shall be transformed into zombie processes, unless SIGCHLD is set to SIG_IGN.

SA_NODEFER If set and *sig* is caught, *sig* shall not be added to the thread's signal mask on entry to the signal handler unless it is included in *sa_mask*. Otherwise, *sig* shall always be added to the thread's signal mask on entry to the signal handler.

When a signal is caught by a signal-catching function installed by *sigaction()*, a new signal mask is calculated and installed for the duration of the signal-catching function (or until a call to either *sigprocmask()* or *sigsuspend()* is made). This mask is formed by taking the union of the current signal mask and the value of the *sa_mask* for the signal being delivered, and unless SA_NODEFER or SA_RESETHAND is set, then including the signal being delivered. If and when the user's signal handler returns normally, the original signal mask is restored.

Once an action is installed for a specific signal, it shall remain installed until another action is explicitly requested (by another call to *sigaction()*), until the SA_RESETHAND flag causes resetting of the handler, or until one of the *exec* functions is called.

If the previous action for *sig* had been established by *signal()*, the values of the fields returned in the structure pointed to by *oact* are unspecified, and in particular *oact->sa_handler* is not necessarily the same value passed to *signal()*. However, if a pointer to the same structure or a copy thereof is passed to a subsequent call to *sigaction()* via the *act* argument, handling of the signal shall be as if the original call to *signal()* were repeated.

If *sigaction()* fails, no new signal handler is installed.

It is unspecified whether an attempt to set the action for a signal that cannot be caught or ignored to SIG_DFL is ignored or causes an error to be returned with *errno* set to [EINVAL].

If SA_SIGINFO is not set in *sa_flags*, then the disposition of subsequent occurrences of *sig* when it is already pending is implementation-defined; the signal-catching function shall be invoked with a single argument. If SA_SIGINFO is set in *sa_flags*, then subsequent occurrences of *sig* generated by *sigqueue()* or as a result of any signal-generating function that supports the specification of an application-defined value (when *sig* is already pending) shall be queued in FIFO order until delivered or accepted; the signal-catching function shall be invoked with three arguments. The application specified value is passed to the signal-catching function as the *si_value* member of the **siginfo_t** structure.

The result of the use of *sigaction()* and a *sigwait()* function concurrently within a process on the same signal is unspecified.

RETURN VALUE

Upon successful completion, *sigaction()* shall return 0; otherwise, -1 shall be returned, *errno* shall be set to indicate the error, and no new signal-catching function shall be installed.

sigaction()*System Interfaces***ERRORS**

The *sigaction()* function shall fail if:

[EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a signal that cannot be caught or ignore a signal that cannot be ignored.

[ENOTSUP] The SA_SIGINFO bit flag is set in the *sa_flags* field of the **sigaction** structure.

The *sigaction()* function may fail if:

[EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be caught or ignored (or both).

In addition, the *sigaction()* function may fail if the SA_SIGINFO flag is set in the *sa_flags* field of the **sigaction** structure for a signal not in the range SIGRTMIN to SIGRTMAX.

EXAMPLES**Establishing a Signal Handler**

The following example demonstrates the use of *sigaction()* to establish a handler for the SIGINT signal.

```
#include <signal.h>

static void handler(int signum)
{
    /* Take appropriate actions for signal delivery */
}

int main()
{
    struct sigaction sa;

    sa.sa_handler = handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = SA_RESTART; /* Restart functions if
                               interrupted by handler */
    if (sigaction(SIGINT, &sa, NULL) == -1)
        /* Handle error */;

    /* Further code */
}
```

APPLICATION USAGE

The *sigaction()* function supersedes the *signal()* function, and should be used in preference. In particular, *sigaction()* and *signal()* should not be used in the same process to control the same signal. The behavior of async-signal-safe functions, as defined in their respective DESCRIPTION sections, is as specified by this volume of POSIX.1-2008, regardless of invocation from a signal-catching function. This is the only intended meaning of the statement that async-signal-safe functions may be used in signal-catching functions without restrictions. Applications must still consider all effects of such functions on such things as data structures, files, and process state. In particular, application developers need to consider the restrictions on interactions when interrupting *sleep()* and interactions among multiple handles for a file description. The fact that any specific function is listed as async-signal-safe does not necessarily mean that invocation of that function from a signal-catching function is recommended.

In order to prevent errors arising from interrupting non-async-signal-safe function calls, applications should protect calls to these functions either by blocking the appropriate signals or

through the use of some programmatic semaphore (see *semget()*, *sem_init()*, *sem_open()*, and so on). Note in particular that even the “safe” functions may modify *errno*; the signal-catching function, if not executing as an independent thread, should save and restore its value in order to avoid the possibility that delivery of a signal in between an error return from a function that sets *errno* and the subsequent examination of *errno* could result in the signal-catching function changing the value of *errno*. Naturally, the same principles apply to the async-signal-safety of application routines and asynchronous data access. Note that *longjmp()* and *siglongjmp()* are not in the list of async-signal-safe functions. This is because the code executing after *longjmp()* and *siglongjmp()* can call any unsafe functions with the same danger as calling those unsafe functions directly from the signal handler. Applications that use *longjmp()* and *siglongjmp()* from within signal handlers require rigorous protection in order to be portable. Many of the other functions that are excluded from the list are traditionally implemented using either *malloc()* or *free()* functions or the standard I/O library, both of which traditionally use data structures in a non-async-signal-safe manner. Since any combination of different functions using a common data structure can cause async-signal-safety problems, this volume of POSIX.1-2008 does not define the behavior when any unsafe function is called in a signal handler that interrupts an unsafe function.

If the signal occurs other than as the result of calling *abort()*, *kill()*, or *raise()*, the behavior is undefined if the signal handler calls any function in the standard library other than one of the functions listed in the table of async-signal-safe functions in Section 2.4.3 (on page 486), or refers to any object other than *errno* with static storage duration other than by assigning a value to a static storage duration variable of type **volatile sig_atomic_t**. Unless all signal handlers have *errno* set on return as it was on entry, the value of *errno* is unspecified.

Usually, the signal is executed on the stack that was in effect before the signal was delivered. An alternate stack may be specified to receive a subset of the signals being caught.

When the signal handler returns, the receiving thread resumes execution at the point it was interrupted unless the signal handler makes other arrangements. If *longjmp()* or *_longjmp()* is used to leave the signal handler, then the signal mask must be explicitly restored.

This volume of POSIX.1-2008 defines the third argument of a signal handling function when SA_SIGINFO is set as a **void *** instead of a **ucontext_t ***, but without requiring type checking. New applications should explicitly cast the third argument of the signal handling function to **ucontext_t ***.

The BSD optional four argument signal handling function is not supported by this volume of POSIX.1-2008. The BSD declaration would be:

```
void handler(int sig, int code, struct sigcontext *scp,
             char *addr);
```

where *sig* is the signal number, *code* is additional information on certain signals, *scp* is a pointer to the **sigcontext** structure, and *addr* is additional address information. Much the same information is available in the objects pointed to by the second argument of the signal handler specified when SA_SIGINFO is set.

Since the *sigaction()* function is allowed but not required to set SA_NODEFER when the application sets the SA_RESETHAND flag, applications which depend on the SA_RESETHAND functionality for the newly installed signal handler must always explicitly set SA_NODEFER when they set SA_RESETHAND in order to be portable.

sigaction()*System Interfaces***RATIONALE**

Although this volume of POSIX.1-2008 requires that signals that cannot be ignored shall not be added to the signal mask when a signal-catching function is entered, there is no explicit requirement that subsequent calls to *sigaction()* reflect this in the information returned in the *oact* argument. In other words, if SIGKILL is included in the *sa_mask* field of *act*, it is unspecified whether or not a subsequent call to *sigaction()* returns with SIGKILL included in the *sa_mask* field of *oact*.

The SA_NOCLDSTOP flag, when supplied in the *act->sa_flags* parameter, allows overloading SIGCHLD with the System V semantics that each SIGCHLD signal indicates a single terminated child. Most conforming applications that catch SIGCHLD are expected to install signal-catching functions that repeatedly call the *waitpid()* function with the WNOHANG flag set, acting on each child for which status is returned, until *waitpid()* returns zero. If stopped children are not of interest, the use of the SA_NOCLDSTOP flag can prevent the overhead from invoking the signal-catching routine when they stop.

Some historical implementations also define other mechanisms for stopping processes, such as the *ptrace()* function. These implementations usually do not generate a SIGCHLD signal when processes stop due to this mechanism; however, that is beyond the scope of this volume of POSIX.1-2008.

This volume of POSIX.1-2008 requires that calls to *sigaction()* that supply a NULL *act* argument succeed, even in the case of signals that cannot be caught or ignored (that is, SIGKILL or SIGSTOP). The System V *signal()* and BSD *sigvec()* functions return [EINVAL] in these cases and, in this respect, their behavior varies from *sigaction()*.

This volume of POSIX.1-2008 requires that *sigaction()* properly save and restore a signal action set up by the ISO C standard *signal()* function. However, there is no guarantee that the reverse is true, nor could there be given the greater amount of information conveyed by the **sigaction** structure. Because of this, applications should avoid using both functions for the same signal in the same process. Since this cannot always be avoided in case of general-purpose library routines, they should always be implemented with *sigaction()*.

It was intended that the *signal()* function should be implementable as a library routine using *sigaction()*.

The POSIX Realtime Extension extends the *sigaction()* function as specified by the POSIX.1-1990 standard to allow the application to request on a per-signal basis via an additional signal action flag that the extra parameters, including the application-defined signal value, if any, be passed to the signal-catching function.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 484), *exec*, *kill()*, *_longjmp()*, *longjmp()*, *pthread_sigmask()*, *raise()*, *semget()*, *sem_init()*, *sem_open()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *signal()*, *sigsuspend()*, *wait()*, *waitid()*

XBD <signal.h>

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and POSIX Threads Extension.

In the DESCRIPTION, the second argument to *func* when SA_SIGINFO is set is no longer permitted to be NULL, and the description of permitted **siginfo_t** contents is expanded by reference to **<signal.h>**.

Since the X/OPEN UNIX Extension functionality is now folded into the BASE, the [ENOTSUP] error is deleted.

Issue 6

The Open Group Corrigendum U028/7 is applied. In the paragraph entitled “Signal Effects on Other Functions”, a reference to *sigpending()* is added.

In the DESCRIPTION, the text “Signal Generation and Delivery”, “Signal Actions”, and “Signal Effects on Other Functions” are moved to a separate section of this volume of POSIX.1-2008.

Text describing functionality from the Realtime Signals Extension option is marked.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The [ENOTSUP] error condition is added.

The normative text is updated to avoid use of the term “must” for application requirements.

The **restrict** keyword is added to the *sigaction()* prototype for alignment with the ISO/IEC 9899: 1999 standard.

References to the *wait3()* function are removed.

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/57 is applied, changing text in the table describing the **sigaction** structure.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/127 is applied, removing text from the DESCRIPTION duplicated later in the same section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/128 is applied, updating the DESCRIPTION and APPLICATION USAGE sections. Changes are made to refer to the thread rather than the process.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/129 is applied, adding the example to the EXAMPLES section.

Issue 7

Austin Group Interpretation 1003.1-2001 #004 is applied.

Austin Group Interpretations 1003.1-2001 #065 and #084 are applied, clarifying the role of the SA_NODEFER flag with respect to the signal mask, and clarifying the SA_RESTART flag for interrupted functions which use timeouts.

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XSH-ERN-167 is applied, updating the APPLICATION USAGE section.

SD5-XSH-ERN-172 is applied, updating the DESCRIPTION to make optional the requirement that when the SA_RESETHAND flag is set, *sigaction()* shall behave as if the SA_NODEFER flag were also set.

sigaction()*System Interfaces*

- 61149 Functionality relating to the Realtime Signals Extension option is moved to the Base.
- 61150 The description of the *si_code* member is replaced with a reference to [Section 2.4.3](#) (on page 486).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

61151 NAME

61152 `sigaddset` — add a signal to a signal set

61153 SYNOPSIS

```
61154 CX #include <signal.h>
61155 int sigaddset(sigset_t *set, int signo);
```

61156 DESCRIPTION

61157 The `sigaddset()` function adds the individual signal specified by the *signo* to the signal set pointed
61158 to by *set*.

61159 Applications shall call either `sigemptyset()` or `sigfillset()` at least once for each object of type
61160 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
61161 nonetheless supplied as an argument to any of `pthread_sigmask()`, `sigaction()`, `sigaddset()`,
61162 `sigdelset()`, `sigismember()`, `sigpending()`, `sigprocmask()`, `sigsuspend()`, `sigtimedwait()`, `sigwait()`, or
61163 `sigwaitinfo()`, the results are undefined.

61164 RETURN VALUE

61165 Upon successful completion, `sigaddset()` shall return 0; otherwise, it shall return `-1` and set *errno*
61166 to indicate the error.

61167 ERRORS

61168 The `sigaddset()` function may fail if:

61169 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

61170 EXAMPLES

61171 None.

61172 APPLICATION USAGE

61173 None.

61174 RATIONALE

61175 None.

61176 FUTURE DIRECTIONS

61177 None.

61178 SEE ALSO

61179 Section 2.4 (on page 484), `pthread_sigmask()`, `sigaction()`, `sigdelset()`, `sigemptyset()`, `sigfillset()`,
61180 `sigismember()`, `sigpending()`, `sigsuspend()`

61181 XBD `<signal.h>`

61182 CHANGE HISTORY

61183 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61184 Issue 5

61185 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
61186 previous issues.

61187 Issue 6

61188 The normative text is updated to avoid use of the term “must” for application requirements.

61189 The SYNOPSIS is marked CX since the presence of this function in the `<signal.h>` header is an
61190 extension over the ISO C standard.

sigaltstack()

System Interfaces

61191 **NAME**

61192 sigaltstack — set and get signal alternate stack context

61193 **SYNOPSIS**

```
61194 XSI #include <signal.h>
61195 int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);
```

61196 **DESCRIPTION**

61197 The *sigaltstack()* function allows a process to define and examine the state of an alternate stack
 61198 for signal handlers for the current thread. Signals that have been explicitly declared to execute
 61199 on the alternate stack shall be delivered on the alternate stack.

61200 If *ss* is not a null pointer, it points to a **stack_t** structure that specifies the alternate signal stack
 61201 that shall take effect upon return from *sigaltstack()*. The *ss_flags* member specifies the new stack
 61202 state. If it is set to *SS_DISABLE*, the stack is disabled and *ss_sp* and *ss_size* are ignored.
 61203 Otherwise, the stack shall be enabled, and the *ss_sp* and *ss_size* members specify the new address
 61204 and size of the stack.

61205 The range of addresses starting at *ss_sp* up to but not including *ss_sp+ss_size* is available to the
 61206 implementation for use as the stack. This function makes no assumptions regarding which end
 61207 is the stack base and in which direction the stack grows as items are pushed.

61208 If *oss* is not a null pointer, upon successful completion it shall point to a **stack_t** structure that
 61209 specifies the alternate signal stack that was in effect prior to the call to *sigaltstack()*. The *ss_sp*
 61210 and *ss_size* members specify the address and size of that stack. The *ss_flags* member specifies the
 61211 stack's state, and may contain one of the following values:

61212 **SS_ONSTACK** The process is currently executing on the alternate signal stack. Attempts to
 61213 modify the alternate signal stack while the process is executing on it fail. This
 61214 flag shall not be modified by processes.

61215 **SS_DISABLE** The alternate signal stack is currently disabled.

61216 The value *SIGSTKSZ* is a system default specifying the number of bytes that would be used to
 61217 cover the usual case when manually allocating an alternate stack area. The value *MINSIGSTKSZ*
 61218 is defined to be the minimum stack size for a signal handler. In computing an alternate stack
 61219 size, a program should add that amount to its stack requirements to allow for the system
 61220 implementation overhead. The constants *SS_ONSTACK*, *SS_DISABLE*, *SIGSTKSZ*, and
 61221 *MINSIGSTKSZ* are defined in **<signal.h>**.

61222 After a successful call to one of the *exec* functions, there are no alternate signal stacks in the new
 61223 process image.

61224 In some implementations, a signal (whether or not indicated to execute on the alternate stack)
 61225 shall always execute on the alternate stack if it is delivered while another signal is being caught
 61226 using the alternate stack.

61227 Use of this function by library threads that are not bound to kernel-scheduled entities results in
 61228 undefined behavior.

61229 **RETURN VALUE**

61230 Upon successful completion, *sigaltstack()* shall return 0; otherwise, it shall return -1 and set *errno*
 61231 to indicate the error.

ERRORS

The *sigaltstack()* function shall fail if:

- [EINVAL] The *ss* argument is not a null pointer, and the *ss_flags* member pointed to by *ss* contains flags other than *SS_DISABLE*.
- [ENOMEM] The size of the alternate stack area is less than *MINSIGSTKSZ*.
- [EPERM] An attempt was made to modify an active stack.

EXAMPLES**Allocating Memory for an Alternate Stack**

The following example illustrates a method for allocating memory for an alternate stack.

```
#include <signal.h>
...
if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
    /* Error return. */
sigstk.ss_size = SIGSTKSZ;
sigstk.ss_flags = 0;
if (sigaltstack(&sigstk, (stack_t *)0) < 0)
    perror("sigaltstack");
```

APPLICATION USAGE

On some implementations, stack space is automatically extended as needed. On those implementations, automatic extension is typically not available for an alternate stack. If the stack overflows, the behavior is undefined.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 484), *exec*, *sigaction()*, *sigsetjmp()*

XBD <signal.h>

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The last sentence of the DESCRIPTION was included as an APPLICATION USAGE note in previous issues.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

The **restrict** keyword is added to the *sigaltstack()* prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/58 is applied, updating the first sentence to include “for the current thread”.

sigdelset()*System Interfaces*61272 **NAME**

61273 sigdelset — delete a signal from a signal set

61274 **SYNOPSIS**

```
61275 CX #include <signal.h>
61276 int sigdelset(sigset_t *set, int signo);
```

61277 **DESCRIPTION**

61278 The *sigdelset()* function deletes the individual signal specified by *signo* from the signal set
 61279 pointed to by *set*.

61280 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type
 61281 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
 61282 nonetheless supplied as an argument to any of *pthread_sigmask()*, *sigaction()*, *sigaddset()*,
 61283 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or
 61284 *sigwaitinfo()*, the results are undefined.

61285 **RETURN VALUE**

61286 Upon successful completion, *sigdelset()* shall return 0; otherwise, it shall return -1 and set *errno*
 61287 to indicate the error.

61288 **ERRORS**61289 The *sigdelset()* function may fail if:

61290 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal
 61291 number.

61292 **EXAMPLES**

61293 None.

61294 **APPLICATION USAGE**

61295 None.

61296 **RATIONALE**

61297 None.

61298 **FUTURE DIRECTIONS**

61299 None.

61300 **SEE ALSO**

61301 Section 2.4 (on page 484), *pthread_sigmask()*, *sigaction()*, *sigaddset()*, *sigemptyset()*, *sigfillset()*,
 61302 *sigismember()*, *sigpending()*, *sigsuspend()*

61303 XBD [<signal.h>](#)61304 **CHANGE HISTORY**

61305 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61306 **Issue 5**

61307 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
 61308 previous issues.

61309 **Issue 6**

61310 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
 61311 extension over the ISO C standard.

NAME

sigemptyset — initialize and empty a signal set

SYNOPSIS

```
#include <signal.h>

int sigemptyset(sigset_t *set);
```

DESCRIPTION

The *sigemptyset()* function initializes the signal set pointed to by *set*, such that all signals defined in POSIX.1-2008 are excluded.

RETURN VALUE

Upon successful completion, *sigemptyset()* shall return 0; otherwise, it shall return -1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The implementation of the *sigemptyset()* (or *sigfillset()*) function could quite trivially clear (or set) all the bits in the signal set. Alternatively, it would be reasonable to initialize part of the structure, such as a version field, to permit binary-compatibility between releases where the size of the set varies. For such reasons, either *sigemptyset()* or *sigfillset()* must be called prior to any other use of the signal set, even if such use is read-only (for example, as an argument to *sigpending()*). This function is not intended for dynamic allocation.

The *sigfillset()* and *sigemptyset()* functions require that the resulting signal set include (or exclude) all the signals defined in this volume of POSIX.1-2008. Although it is outside the scope of this volume of POSIX.1-2008 to place this requirement on signals that are implemented as extensions, it is recommended that implementation-defined signals also be affected by these functions. However, there may be a good reason for a particular signal not to be affected. For example, blocking or ignoring an implementation-defined signal may have undesirable side-effects, whereas the default action for that signal is harmless. In such a case, it would be preferable for such a signal to be excluded from the signal set returned by *sigfillset()*.

In early proposals there was no distinction between invalid and unsupported signals (the names of optional signals that were not supported by an implementation were not defined by that implementation). The [EINVAL] error was thus specified as a required error for invalid signals. With that distinction, it is not necessary to require implementations of these functions to determine whether an optional signal is actually supported, as that could have a significant performance impact for little value. The error could have been required for invalid signals and optional for unsupported signals, but this seemed unnecessarily complex. Thus, the error is optional in both cases.

FUTURE DIRECTIONS

None.

sigemptyset()*System Interfaces*61354 **SEE ALSO**

61355 Section 2.4 (on page 484), *pthread_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*,
61356 *sigismember()*, *sigpending()*, *sigsuspend()*

61357 XBD **<signal.h>**

61358 **CHANGE HISTORY**

61359 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61360 **Issue 6**

61361 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
61362 extension over the ISO C standard.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

sigfillset — initialize and fill a signal set

SYNOPSIS

```
#include <signal.h>

int sigfillset(sigset_t *set);
```

DESCRIPTION

The *sigfillset()* function shall initialize the signal set pointed to by *set*, such that all signals defined in this volume of POSIX.1-2008 are included.

RETURN VALUE

Upon successful completion, *sigfillset()* shall return 0; otherwise, it shall return -1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *sigemptyset()* (on page 1927).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 484), *pthread_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigismember()*, *sigpending()*, *sigsuspend()*

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

sighold()

System Interfaces

61393 **NAME**61394 **sighold, sigignore, sigpause, sigrelse, sigset** — signal management61395 **SYNOPSIS**

```

61396 OB XSI #include <signal.h>
61397
61397 int sighold(int sig);
61398 int sigignore(int sig);
61399 int sigpause(int sig);
61400 int sigrelse(int sig);
61401 void (*sigset(int sig, void (*disp)(int)))(int);

```

61402 **DESCRIPTION**

61403 Use of any of these functions is unspecified in a multi-threaded process.

61404 The *sighold()*, *sigignore()*, *sigpause()*, *sigrelse()*, and *sigset()* functions provide simplified signal management.

61406 The *sigset()* function shall modify signal dispositions. The *sig* argument specifies the signal, which may be any signal except SIGKILL and SIGSTOP. The *disp* argument specifies the signal's disposition, which may be SIG_DFL, SIG_IGN, or the address of a signal handler. If *sigset()* is used, and *disp* is the address of a signal handler, the system shall add *sig* to the signal mask of the calling process before executing the signal handler; when the signal handler returns, the system shall restore the signal mask of the calling process to its state prior to the delivery of the signal. In addition, if *sigset()* is used, and *disp* is equal to SIG_HOLD, *sig* shall be added to the signal mask of the calling process and *sig*'s disposition shall remain unchanged. If *sigset()* is used, and *disp* is not equal to SIG_HOLD, *sig* shall be removed from the signal mask of the calling process.

61416 The *sighold()* function shall add *sig* to the signal mask of the calling process.61417 The *sigrelse()* function shall remove *sig* from the signal mask of the calling process.61418 The *sigignore()* function shall set the disposition of *sig* to SIG_IGN.

61419 The *sigpause()* function shall remove *sig* from the signal mask of the calling process and suspend the calling process until a signal is received. The *sigpause()* function shall restore the signal mask of the process to its original state before returning.

61422 If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling processes shall not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited-for children that were transformed into zombie processes, it shall block until all of its children terminate, and *wait()*, *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD].

61427 **RETURN VALUE**

61428 Upon successful completion, *sigset()* shall return SIG_HOLD if the signal had been blocked and the signal's previous disposition if it had not been blocked. Otherwise, SIG_ERR shall be returned and *errno* set to indicate the error.

61431 The *sigpause()* function shall suspend execution of the thread until a signal is received, whereupon it shall return -1 and set *errno* to [EINTR].

61433 For all other functions, upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

These functions shall fail if:

[EINVAL] The *sig* argument is an illegal signal number.

The *sigset()* and *sigignore()* functions shall fail if:

[EINVAL] An attempt is made to catch a signal that cannot be caught, or to ignore a signal that cannot be ignored.

EXAMPLES

None.

APPLICATION USAGE

The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling signals; new applications should use the *sigaction()* function instead of the obsolescent *sigset()* function.

The *sighold()* function, in conjunction with *sigrelse()* or *sigpause()*, may be used to establish critical regions of code that require the delivery of a signal to be temporarily deferred. For broader portability, the *pthread_sigmask()* or *sigprocmask()* functions should be used instead of the obsolescent *sighold()* and *sigrelse()* functions.

For broader portability, the *sigsuspend()* function should be used instead of the obsolescent *sigpause()* function.

RATIONALE

Each of these historic functions has a direct analog in the other functions which are required to be per-thread and thread-safe (aside from *sigprocmask()*, which is replaced by *pthread_sigmask()*). The *sigset()* function can be implemented as a simple wrapper for *sigaction()*. The *sighold()* function is equivalent to *sigprocmask()* or *pthread_sigmask()* with SIG_BLOCK set. The *sigignore()* function is equivalent to *sigaction()* with SIG_IGN set. The *sigpause()* function is equivalent to *sigsuspend()*. The *sigrelse()* function is equivalent to *sigprocmask()* or *pthread_sigmask()* with SIG_UNBLOCK set.

FUTURE DIRECTIONS

These functions may be removed in a future version.

SEE ALSO

Section 2.4 (on page 484), *exec*, *pause()*, *pthread_sigmask()*, *sigaction()*, *signal()*, *sigsuspend()*, *wait()*, *waitid()*

XBD <signal.h>

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The DESCRIPTION is updated to indicate that the *sigpause()* function restores the signal mask of the process to its original state before returning.

The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to [EINTR].

sighold()*System Interfaces*61476 **Issue 6**

61477 The normative text is updated to avoid use of the term “must” for application requirements.

61478 References to the *wait3()* function are removed.

61479 The XSI functions are split out into their own reference page.

61480 **Issue 7**61481 SD5-XSH-ERN-113 and SD5-XSH-ERN-42 are applied, marking these functions obsolescent and
61482 updating the APPLICATION USAGE and RATIONALE sections.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

siginterrupt — allow signals to interrupt functions

SYNOPSIS

```
#include <signal.h>

int siginterrupt(int sig, int flag);
```

DESCRIPTION

The *siginterrupt()* function shall change the restart behavior when a function is interrupted by the specified signal. The function *siginterrupt(sig, flag)* has an effect as if implemented as:

```
int siginterrupt(int sig, int flag) {
    int ret;
    struct sigaction act;

    (void) sigaction(sig, NULL, &act);
    if (flag)
        act.sa_flags &= ~SA_RESTART;
    else
        act.sa_flags |= SA_RESTART;
    ret = sigaction(sig, &act, NULL);
    return ret;
}
```

RETURN VALUE

Upon successful completion, *siginterrupt()* shall return 0; otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *siginterrupt()* function shall fail if:

[EINVAL] The *sig* argument is not a valid signal number.

EXAMPLES

None.

APPLICATION USAGE

The *siginterrupt()* function supports programs written to historical system interfaces. Applications should use the *sigaction()* with the SA_RESTART flag instead of the obsolescent *siginterrupt()* function.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 484), *sigaction()*

XBD <signal.h>

CHANGE HISTORY

First released in Issue 4, Version 2.

siginterrupt()*System Interfaces*61523 **Issue 5**

61524 Moved from X/OPEN UNIX extension to BASE.

61525 **Issue 6**61526 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/59 is applied, correcting the declaration in
61527 the sample implementation given in the DESCRIPTION.61528 **Issue 7**61529 The *siginterrupt()* function is marked obsolescent.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

61530 NAME

61531 **sigismember** — test for a signal in a signal set

61532 SYNOPSIS

```
61533 CX #include <signal.h>
61534 int sigismember(const sigset_t *set, int signo);
```

61535 DESCRIPTION

61536 The *sigismember()* function shall test whether the signal specified by *signo* is a member of the set
61537 pointed to by *set*.

61538 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type
61539 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
61540 nonetheless supplied as an argument to any of *pthread_sigmask()*, *sigaction()*, *sigaddset()*,
61541 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or
61542 *sigwaitinfo()*, the results are undefined.

61543 RETURN VALUE

61544 Upon successful completion, *sigismember()* shall return 1 if the specified signal is a member of
61545 the specified set, or 0 if it is not. Otherwise, it shall return -1 and set *errno* to indicate the error.

61546 ERRORS

61547 The *sigismember()* function may fail if:

61548 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal
61549 number.

61550 EXAMPLES

61551 None.

61552 APPLICATION USAGE

61553 None.

61554 RATIONALE

61555 None.

61556 FUTURE DIRECTIONS

61557 None.

61558 SEE ALSO

61559 Section 2.4 (on page 484), *pthread_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*,
61560 *sigemptyset()*, *sigpending()*, *sigsuspend()*

61561 XBD [<signal.h>](#)

61562 CHANGE HISTORY

61563 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61564 Issue 5

61565 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
61566 previous issues.

61567 Issue 6

61568 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
61569 extension over the ISO C standard.

siglongjmp()

System Interfaces

61570 **NAME**

61571 siglongjmp — non-local goto with signal handling

61572 **SYNOPSIS**

```
61573 CX #include <setjmp.h>
61574 void siglongjmp(sigjmp_buf env, int val);
```

61575 **DESCRIPTION**61576 The *siglongjmp()* function shall be equivalent to the *longjmp()* function, except as follows:

- 61577 • References to *setjmp()* shall be equivalent to *sigsetjmp()*.
- 61578 • The *siglongjmp()* function shall restore the saved signal mask if and only if the *env*
- 61579 argument was initialized by a call to *sigsetjmp()* with a non-zero *savemask* argument.

61580 **RETURN VALUE**

61581 After *siglongjmp()* is completed, program execution shall continue as if the corresponding

61582 invocation of *sigsetjmp()* had just returned the value specified by *val*. The *siglongjmp()* function

61583 shall not cause *sigsetjmp()* to return 0; if *val* is 0, *sigsetjmp()* shall return the value 1.

61584 **ERRORS**

61585 No errors are defined.

61586 **EXAMPLES**

61587 None.

61588 **APPLICATION USAGE**

61589 The distinction between *setjmp()* or *longjmp()* and *sigsetjmp()* or *siglongjmp()* is only significant

61590 for programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

61591 **RATIONALE**

61592 None.

61593 **FUTURE DIRECTIONS**

61594 None.

61595 **SEE ALSO**61596 *longjmp()*, *pthread_sigmask()*, *setjmp()*, *sigsetjmp()*, *sigsuspend()*61597 XBD **<setjmp.h>**61598 **CHANGE HISTORY**

61599 First released in Issue 3. Included for alignment with the ISO POSIX-1 standard.

61600 **Issue 5**

61601 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

61602 **Issue 6**61603 The DESCRIPTION is rewritten in terms of *longjmp()*.

61604 The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an

61605 extension over the ISO C standard.

NAME

signal — signal management

SYNOPSIS

```
#include <signal.h>

void (*signal(int sig, void (*func)(int)))(int);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

Use of this function is unspecified in a multi-threaded process.

The *signal()* function chooses one of three ways in which receipt of the signal number *sig* is to be subsequently handled. If the value of *func* is *SIG_DFL*, default handling for that signal shall occur. If the value of *func* is *SIG_IGN*, the signal shall be ignored. Otherwise, the application shall ensure that *func* points to a function to be called when that signal occurs. An invocation of such a function because of a signal, or (recursively) of any further functions called by that invocation (other than functions in the standard library), is called a “signal handler”.

When a signal occurs, and *func* points to a function, it is implementation-defined whether the equivalent of a:

```
signal(sig, SIG_DFL);
```

is executed or the implementation prevents some implementation-defined set of signals (at least including *sig*) from occurring until the current signal handling has completed. (If the value of *sig* is *SIGILL*, the implementation may alternatively define that no action is taken.) Next the equivalent of:

```
(*func)(sig);
```

is executed. If and when the function returns, if the value of *sig* was *SIGFPE*, *SIGILL*, or *SIGSEGV* or any other implementation-defined value corresponding to a computational exception, the behavior is undefined. Otherwise, the program shall resume execution at the point it was interrupted. If the signal occurs as the result of calling the *abort()*, *raise()*, *kill()*, *pthread_kill()*, or *sigqueue()* function, the signal handler shall not call the *raise()* function.

If the signal occurs other than as the result of calling *abort()*, *raise()*, *kill()*, *pthread_kill()*, or *sigqueue()*, the behavior is undefined if the signal handler refers to any object with static storage duration other than by assigning a value to an object declared as volatile **sig_atomic_t**, or if the signal handler calls any function in the standard library other than one of the functions listed in [Section 2.4](#) (on page 484). Furthermore, if such a call fails, the value of *errno* is unspecified.

At program start-up, the equivalent of:

```
signal(sig, SIG_IGN);
```

is executed for some signals, and the equivalent of:

```
signal(sig, SIG_DFL);
```

is executed for all other signals (see *exec*).

RETURN VALUE

If the request can be honored, *signal()* shall return the value of *func* for the most recent call to *signal()* for the specified signal *sig*. Otherwise, *SIG_ERR* shall be returned and a positive value shall be stored in *errno*.

signal()*System Interfaces***61649 ERRORS**

61650 The *signal()* function shall fail if:

61651 CX [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a
61652 signal that cannot be caught or ignore a signal that cannot be ignored.

61653 The *signal()* function may fail if:

61654 CX [EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be
61655 caught or ignored (or both).

61656 EXAMPLES

61657 None.

61658 APPLICATION USAGE

61659 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling
61660 signals; new applications should use *sigaction()* rather than *signal()*.

61661 RATIONALE

61662 None.

61663 FUTURE DIRECTIONS

61664 None.

61665 SEE ALSO

61666 Section 2.4 (on page 484), *exec*, *pause()*, *sigaction()*, *sigsuspend()*, *waitid()*

61667 XBD <signal.h>

61668 CHANGE HISTORY

61669 First released in Issue 1. Derived from Issue 1 of the SVID.

61670 Issue 5

61671 Moved from X/OPEN UNIX extension to BASE.

61672 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the signal mask of
61673 the process to its original state before returning.

61674 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends
61675 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to
61676 [EINTR].

61677 Issue 6

61678 Extensions beyond the ISO C standard are marked.

61679 The normative text is updated to avoid use of the term “must” for application requirements.

61680 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

61681 References to the *wait3()* function are removed.

61682 The *sighold()*, *sigignore()*, *sigrelse()*, and *sigset()* functions are split out onto their own reference
61683 page.

61684 **NAME**

61685 signbit — test sign

61686 **SYNOPSIS**

61687 #include <math.h>

61688 int signbit(real-floating x);

61689 **DESCRIPTION**

61690 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 61691 conflict between the requirements described here and the ISO C standard is unintentional. This
 61692 volume of POSIX.1-2008 defers to the ISO C standard.

61693 The *signbit()* macro shall determine whether the sign of its argument value is negative. NaNs,
 61694 zeros, and infinities have a sign bit.

61695 **RETURN VALUE**

61696 The *signbit()* macro shall return a non-zero value if and only if the sign of its argument value is
 61697 negative.

61698 **ERRORS**

61699 No errors are defined.

61700 **EXAMPLES**

61701 None.

61702 **APPLICATION USAGE**

61703 None.

61704 **RATIONALE**

61705 None.

61706 **FUTURE DIRECTIONS**

61707 None.

61708 **SEE ALSO**61709 *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *isnormal()*

61710 XBD <math.h>

61711 **CHANGE HISTORY**

61712 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

signgam()*System Interfaces*61713 **NAME**

61714 signgam — log gamma function

61715 **SYNOPSIS**

```
61716 XSI #include <math.h>  
61717 extern int signgam;
```

61718 **DESCRIPTION**61719 Refer to *lgamma()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***sigpause()**61720 **NAME**

61721 sigpause — remove a signal from the signal mask and suspend the thread

61722 **SYNOPSIS**

```
61723 OB XSI #include <signal.h>  
61724 int sigpause(int sig);
```

61725 **DESCRIPTION**61726 Refer to *sighold()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

sigpending()

System Interfaces

61727 **NAME**

61728 sigpending — examine pending signals

61729 **SYNOPSIS**

```
61730 CX    #include <signal.h>
61731    int sigpending(sigset_t *set);
```

61732 **DESCRIPTION**

61733 The *sigpending()* function shall store, in the location referenced by the *set* argument, the set of
 61734 signals that are blocked from delivery to the calling thread and that are pending on the process
 61735 or the calling thread.

61736 **RETURN VALUE**

61737 Upon successful completion, *sigpending()* shall return 0; otherwise, -1 shall be returned and
 61738 *errno* set to indicate the error.

61739 **ERRORS**

61740 No errors are defined.

61741 **EXAMPLES**

61742 None.

61743 **APPLICATION USAGE**

61744 None.

61745 **RATIONALE**

61746 None.

61747 **FUTURE DIRECTIONS**

61748 None.

61749 **SEE ALSO**

61750 *exec*, *pthread_sigmask()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*

61751 XBD **<signal.h>**

61752 **CHANGE HISTORY**

61753 First released in Issue 3.

61754 **Issue 5**

61755 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

61756 **Issue 6**

61757 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
 61758 extension over the ISO C standard.

*System Interfaces***sigprocmask()**61759 **NAME**

61760 sigprocmask — examine and change blocked signals

61761 **SYNOPSIS**

```
61762 CX #include <signal.h>
61763      int sigprocmask(int how, const sigset_t *restrict set,
61764                      sigset_t *restrict oset);
```

61765 **DESCRIPTION**61766 Refer to *pthread_sigmask()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

sigqueue()*System Interfaces*61767 **NAME**

61768 sigqueue — queue a signal to a process

61769 **SYNOPSIS**

```
61770 CX #include <signal.h>
61771 int sigqueue(pid_t pid, int signo, const union sigval value);
```

61772 **DESCRIPTION**

61773 The *sigqueue()* function shall cause the signal specified by *signo* to be sent with the value
 61774 specified by *value* to the process specified by *pid*. If *signo* is zero (the null signal), error checking
 61775 is performed but no signal is actually sent. The null signal can be used to check the validity of
 61776 *pid*.

61777 The conditions required for a process to have permission to queue a signal to another process are
 61778 the same as for the *kill()* function.

61779 The *sigqueue()* function shall return immediately. If SA_SIGINFO is set for *signo* and if the
 61780 resources were available to queue the signal, the signal shall be queued and sent to the receiving
 61781 process. If SA_SIGINFO is not set for *signo*, then *signo* shall be sent at least once to the receiving
 61782 process; it is unspecified whether *value* shall be sent to the receiving process as a result of this
 61783 call.

61784 If the value of *pid* causes *signo* to be generated for the sending process, and if *signo* is not blocked
 61785 for the calling thread and if no other thread has *signo* unblocked or is waiting in a *sigwait()*
 61786 function for *signo*, either *signo* or at least the pending, unblocked signal shall be delivered to the
 61787 calling thread before the *sigqueue()* function returns. Should any multiple pending signals in the
 61788 range SIGRTMIN to SIGRTMAX be selected for delivery, it shall be the lowest numbered one.
 61789 The selection order between realtime and non-realtime signals, or between multiple pending
 61790 non-realtime signals, is unspecified.

61791 **RETURN VALUE**

61792 Upon successful completion, the specified signal shall have been queued, and the *sigqueue()*
 61793 function shall return a value of zero. Otherwise, the function shall return a value of -1 and set
 61794 *errno* to indicate the error.

61795 **ERRORS**

61796 The *sigqueue()* function shall fail if:

61797 61798 61799	[EAGAIN]	No resources are available to queue the signal. The process has already queued {SIGQUEUE_MAX} signals that are still pending at the receiver(s), or a system-wide resource limit has been exceeded.
61800	[EINVAL]	The value of the <i>signo</i> argument is an invalid or unsupported signal number.
61801 61802	[EPERM]	The process does not have appropriate privileges to send the signal to the receiving process.
61803	[ESRCH]	The process <i>pid</i> does not exist.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *sigqueue()* function allows an application to queue a realtime signal to itself or to another process, specifying the application-defined value. This is common practice in realtime applications on existing realtime systems. It was felt that specifying another function in the *sig...* name space already carved out for signals was preferable to extending the interface to *kill()*.

Such a function became necessary when the put/get event function of the message queues was removed. It should be noted that the *sigqueue()* function implies reduced performance in a security-conscious implementation as the access permissions between the sender and receiver have to be checked on each send when the *pid* is resolved into a target process. Such access checks were necessary only at message queue open in the previous interface.

The standard developers required that *sigqueue()* have the same semantics with respect to the null signal as *kill()*, and that the same permission checking be used. But because of the difficulty of implementing the “broadcast” semantic of *kill()* (for example, to process groups) and the interaction with resource allocation, this semantic was not adopted. The *sigqueue()* function queues a signal to a single process specified by the *pid* argument.

The *sigqueue()* function can fail if the system has insufficient resources to queue the signal. An explicit limit on the number of queued signals that a process could send was introduced. While the limit is “per-sender”, this volume of POSIX.1-2008 does not specify that the resources be part of the state of the sender. This would require either that the sender be maintained after exit until all signals that it had sent to other processes were handled or that all such signals that had not yet been acted upon be removed from the queue(s) of the receivers. This volume of POSIX.1-2008 does not preclude this behavior, but an implementation that allocated queuing resources from a system-wide pool (with per-sender limits) and that leaves queued signals pending after the sender exits is also permitted.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.8.1 (on page 497)

XBD <signal.h>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6

The *sigqueue()* function is marked as part of the Realtime Signals Extension option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Realtime Signals Extension option.

Issue 7

The *sigqueue()* function is moved from the Realtime Signals Extension option to the Base.

sigrelse()*System Interfaces*61847 **NAME**

61848 sigrelse, sigset — signal management

61849 **SYNOPSIS**

```
61850 OB XSI #include <signal.h>
61851 int sigrelse(int sig);
61852 void (*sigset(int sig, void (*disp)(int)))(int);
```

61853 **DESCRIPTION**61854 Refer to *sighold()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

sigsetjmp — set jump point for a non-local goto

SYNOPSIS

```
#include <setjmp.h>

int sigsetjmp(sigjmp_buf env, int savemask);
```

DESCRIPTION

The *sigsetjmp()* function shall be equivalent to the *setjmp()* function, except as follows:

- References to *setjmp()* are equivalent to *sigsetjmp()*.
- References to *longjmp()* are equivalent to *siglongjmp()*.
- If the value of the *savemask* argument is not 0, *sigsetjmp()* shall also save the current signal mask of the calling thread as part of the calling environment.

RETURN VALUE

If the return is from a successful direct invocation, *sigsetjmp()* shall return 0. If the return is from a call to *siglongjmp()*, *sigsetjmp()* shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The distinction between *setjmp()/longjmp()* and *sigsetjmp()/siglongjmp()* is only significant for programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

Note that since this function is defined in terms of *setjmp()*, if *savemask* is zero, it is unspecified whether the signal mask is saved.

RATIONALE

The ISO C standard specifies various restrictions on the usage of the *setjmp()* macro in order to permit implementors to recognize the name in the compiler and not implement an actual function. These same restrictions apply to the *sigsetjmp()* macro.

There are processors that cannot easily support these calls, but this was not considered a sufficient reason to exclude them.

4.2 BSD, 4.3 BSD, and XSI-conformant systems provide functions named *_setjmp()* and *_longjmp()* that, together with *setjmp()* and *longjmp()*, provide the same functionality as *sigsetjmp()* and *siglongjmp()*. On those systems, *setjmp()* and *longjmp()* save and restore signal masks, while *_setjmp()* and *_longjmp()* do not. On System V Release 3 and in corresponding issues of the SVID, *setjmp()* and *longjmp()* are explicitly defined not to save and restore signal masks. In order to permit existing practice in both cases, the relation of *setjmp()* and *longjmp()* to signal masks is not specified, and a new set of functions is defined instead.

The *longjmp()* and *siglongjmp()* functions operate as in the previous issue provided the matching *setjmp()* or *sigsetjmp()* has been performed in the same thread. Non-local jumps into contexts saved by other threads would be at best a questionable practice and were not considered worthy of standardization.

sigsetjmp()*System Interfaces*61895 **FUTURE DIRECTIONS**

61896 None.

61897 **SEE ALSO**61898 *pthread_sigmask()*, *siglongjmp()*, *signal()*, *sigsuspend()*61899 XBD **<setjmp.h>**61900 **CHANGE HISTORY**

61901 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61902 **Issue 5**

61903 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

61904 **Issue 6**61905 The DESCRIPTION is reworded in terms of *setjmp()*.61906 The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an
61907 extension over the ISO C standard.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

sigsuspend — wait for a signal

SYNOPSIS

```
#include <signal.h>

int sigsuspend(const sigset_t *sigmask);
```

DESCRIPTION

The *sigsuspend()* function shall replace the current signal mask of the calling thread with the set of signals pointed to by *sigmask* and then suspend the thread until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process. This shall not cause any other signals that may have been pending on the process to become pending on the thread.

If the action is to terminate the process then *sigsuspend()* shall never return. If the action is to execute a signal-catching function, then *sigsuspend()* shall return after the signal-catching function returns, with the signal mask restored to the set that existed prior to the *sigsuspend()* call.

It is not possible to block signals that cannot be ignored. This is enforced by the system without causing an error to be indicated.

RETURN VALUE

Since *sigsuspend()* suspends thread execution indefinitely, there is no successful completion return value. If a return occurs, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *sigsuspend()* function shall fail if:

[EINTR] A signal is caught by the calling process and control is returned from the signal-catching function.

EXAMPLES

None.

APPLICATION USAGE

Normally, at the beginning of a critical code section, a specified set of signals is blocked using the *sigprocmask()* function. When the thread has completed the critical section and needs to wait for the previously blocked signal(s), it pauses by calling *sigsuspend()* with the mask that was returned by the *sigprocmask()* call.

RATIONALE

Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers can install an additional cancellation handler which resets the signal mask to the expected value.

```
void cleanup(void *arg)
{
    sigset_t *ss = (sigset_t *) arg;
    pthread_sigmask(SIG_SETMASK, ss, NULL);
}

int call_sigsuspend(const sigset_t *mask)
{
    sigset_t oldmask;
    int result;
    pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
    pthread_cleanup_push(cleanup, &oldmask);
```

sigsuspend()

System Interfaces

```
61953         result = sigsuspend(sigmask);
61954         pthread_cleanup_pop(0);
61955         return result;
61956     }
```

61957 FUTURE DIRECTIONS

61958 None.

61959 SEE ALSO

61960 [Section 2.4](#) (on page 484), [pause\(\)](#), [sigaction\(\)](#), [sigaddset\(\)](#), [sigdelset\(\)](#), [sigemptyset\(\)](#), [sigfillset\(\)](#)

61961 XBD [<signal.h>](#)

61962 CHANGE HISTORY

61963 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61964 Issue 5

61965 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

61966 Issue 6

61967 The text in the RETURN VALUE section has been changed from “suspends process execution”
61968 to “suspends thread execution”. This reflects IEEE PASC Interpretation 1003.1c #40.

61969 Text in the APPLICATION USAGE section has been replaced.

61970 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an
61971 extension over the ISO C standard.

61972 Issue 7

61973 SD5-XSH-ERN-122 is applied, adding the example code in the RATIONALE.

System Interfaces

sigtimedwait()**NAME**

sigtimedwait, sigwaitinfo — wait for queued signals

SYNOPSIS

```
#include <signal.h>

int sigtimedwait(const sigset_t *restrict set,
                 siginfo_t *restrict info,
                 const struct timespec *restrict timeout);
int sigwaitinfo(const sigset_t *restrict set,
                siginfo_t *restrict info);
```

DESCRIPTION

The *sigtimedwait()* function shall be equivalent to *sigwaitinfo()* except that if none of the signals specified by *set* are pending, *sigtimedwait()* shall wait for the time interval specified in the **timespec** structure referenced by *timeout*. If the **timespec** structure pointed to by *timeout* is zero-valued and if none of the signals specified by *set* are pending, then *sigtimedwait()* shall return immediately with an error. If *timeout* is the null pointer, the behavior is unspecified. If the Monotonic Clock option is supported, the **CLOCK_MONOTONIC** clock shall be used to measure the time interval specified by the *timeout* argument.

The *sigwaitinfo()* function selects the pending signal from the set specified by *set*. Should any of multiple pending signals in the range **SIGRTMIN** to **SIGRTMAX** be selected, it shall be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified. If no signal in *set* is pending at the time of the call, the calling thread shall be suspended until one or more signals in *set* become pending or until it is interrupted by an unblocked, caught signal.

The *sigwaitinfo()* function shall be equivalent to the *sigwait()* function if the *info* argument is **NULL**. If the *info* argument is non-**NULL**, the *sigwaitinfo()* function shall be equivalent to *sigwait()*, except that the selected signal number shall be stored in the *si_signo* member, and the cause of the signal shall be stored in the *si_code* member. If any value is queued to the selected signal, the first such queued value shall be dequeued and, if the *info* argument is non-**NULL**, the value shall be stored in the *si_value* member of *info*. The system resource used to queue the signal shall be released and returned to the system for other use. If no value is queued, the content of the *si_value* member is undefined. If no further signals are queued for the selected signal, the pending indication for that signal shall be reset.

RETURN VALUE

Upon successful completion (that is, one of the signals specified by *set* is pending or is generated) *sigwaitinfo()* and *sigtimedwait()* shall return the selected signal number. Otherwise, the function shall return a value of **-1** and set *errno* to indicate the error.

ERRORS

The *sigtimedwait()* function shall fail if:

[EAGAIN] No signal specified by *set* was generated within the specified timeout period.

The *sigtimedwait()* and *sigwaitinfo()* functions may fail if:

[EINTR] The wait was interrupted by an unblocked, caught signal. It shall be documented in system documentation whether this error causes these functions to fail.

sigtimedwait()*System Interfaces*

62017 The *sigtimedwait()* function may also fail if:

62018 [EINVAL] The *timeout* argument specified a *tv_nsec* value less than zero or greater than

62019 or equal to 1 000 million.

62020 An implementation should only check for this error if no signal is pending in *set* and it is

62021 necessary to wait.

EXAMPLES

62022 None.

APPLICATION USAGE

62025 The *sigtimedwait()* function times out and returns an [EAGAIN] error. Application developers

62026 should note that this is inconsistent with other functions such as *pthread_cond_timedwait()* that

62027 return [ETIMEDOUT].

RATIONALE

62029 Existing programming practice on realtime systems uses the ability to pause waiting for a

62030 selected set of events and handle the first event that occurs in-line instead of in a signal-handling

62031 function. This allows applications to be written in an event-directed style similar to a state

62032 machine. This style of programming is useful for largescale transaction processing in which the

62033 overall throughput of an application and the ability to clearly track states are more important

62034 than the ability to minimize the response time of individual event handling.

62035 It is possible to construct a signal-waiting macro function out of the realtime signal function

62036 mechanism defined in this volume of POSIX.1-2008. However, such a macro has to include the

62037 definition of a generalized handler for all signals to be waited on. A significant portion of the

62038 overhead of handler processing can be avoided if the signal-waiting function is provided by the

62039 kernel. This volume of POSIX.1-2008 therefore provides two signal-waiting functions—one that

62040 waits indefinitely and one with a timeout—as part of the overall realtime signal function

62041 specification.

62042 The specification of a function with a timeout allows an application to be written that can be

62043 broken out of a wait after a set period of time if no event has occurred. It was argued that setting

62044 a timer event before the wait and recognizing the timer event in the wait would also implement

62045 the same functionality, but at a lower performance level. Because of the performance

62046 degradation associated with the user-level specification of a timer event and the subsequent

62047 cancellation of that timer event after the wait completes for a valid event, and the complexity

62048 associated with handling potential race conditions associated with the user-level method, the

62049 separate function has been included.

62050 Note that the semantics of the *sigwaitinfo()* function are nearly identical to that of the *sigwait()*

62051 function defined by this volume of POSIX.1-2008. The only difference is that *sigwaitinfo()* returns

62052 the queued signal value in the *value* argument. The return of the queued value is required so that

62053 applications can differentiate between multiple events queued to the same signal number.

62054 The two distinct functions are being maintained because some implementations may choose to

62055 implement the POSIX Threads Extension functions and not implement the queued signals

62056 extensions. Note, though, that *sigwaitinfo()* does not return the queued value if the *value*

62057 argument is NULL, so the POSIX Threads Extension *sigwait()* function can be implemented as a

62058 macro on *sigwaitinfo()*.

62059 The *sigtimedwait()* function was separated from the *sigwaitinfo()* function to address concerns

62060 regarding the overloading of the *timeout* pointer to indicate indefinite wait (no timeout), timed

62061 wait, and immediate return, and concerns regarding consistency with other functions where the

62062 conditional and timed waits were separate functions from the pure blocking function. The

62063 semantics of *sigtimedwait()* are specified such that *sigwaitinfo()* could be implemented as a macro

with a null pointer for *timeout*.

The *sigwait* functions provide a synchronous mechanism for threads to wait for asynchronously-generated signals. One important question was how many threads that are suspended in a call to a *sigwait*() function for a signal should return from the call when the signal is sent. Four choices were considered:

1. Return an error for multiple simultaneous calls to *sigwait* functions for the same signal.
2. One or more threads return.
3. All waiting threads return.
4. Exactly one thread returns.

Prohibiting multiple calls to *sigwait*() for the same signal was felt to be overly restrictive. The “one or more” behavior made implementation of conforming packages easy at the expense of forcing POSIX threads clients to protect against multiple simultaneous calls to *sigwait*() in application code in order to achieve predictable behavior. There was concern that the “all waiting threads” behavior would result in “signal broadcast storms”, consuming excessive CPU resources by replicating the signals in the general case. Furthermore, no convincing examples could be presented that delivery to all was either simpler or more powerful than delivery to one.

Thus, the consensus was that exactly one thread that was suspended in a call to a *sigwait* function for a signal should return when that signal occurs. This is not an onerous restriction as:

- A multi-way signal wait can be built from the single-way wait.
- Signals should only be handled by application-level code, as library routines cannot guess what the application wants to do with signals generated for the entire process.
- Applications can thus arrange for a single thread to wait for any given signal and call any needed routines upon its arrival.

In an application that is using signals for interprocess communication, signal processing is typically done in one place. Alternatively, if the signal is being caught so that process cleanup can be done, the signal handler thread can call separate process cleanup routines for each portion of the application. Since the application main line started each portion of the application, it is at the right abstraction level to tell each portion of the application to clean up.

Certainly, there exist programming styles where it is logical to consider waiting for a single signal in multiple threads. A simple *sigwait_multiple*() routine can be constructed to achieve this goal. A possible implementation would be to have each *sigwait_multiple*() caller registered as having expressed interest in a set of signals. The caller then waits on a thread-specific condition variable. A single server thread calls a *sigwait*() function on the union of all registered signals. When the *sigwait*() function returns, the appropriate state is set and condition variables are broadcast. New *sigwait_multiple*() callers may cause the pending *sigwait*() call to be canceled and reissued in order to update the set of signals being waited for.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.8.1 (on page 497), *pause*(), *pthread_sigmask*(), *sigaction*(), *sigpending*(), *sigsuspend*(), *sigwait*()

XBD *<signal.h>*, *<time.h>*

sigtimedwait()

System Interfaces

62106 **CHANGE HISTORY**

62107 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
 62108 POSIX Threads Extension.

62109 **Issue 6**

62110 These functions are marked as part of the Realtime Signals Extension option.

62111 The Open Group Corrigendum U035/3 is applied. The SYNOPSIS of the *sigwaitinfo()* function
 62112 has been corrected so that the second argument is of type **siginfo_t** *.

62113 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 62114 implementation does not support the Realtime Signals Extension option.

62115 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the
 62116 CLOCK_MONOTONIC clock, if supported, is used to measure timeout intervals.

62117 The **restrict** keyword is added to the *sigtimedwait()* and *sigwaitinfo()* prototypes for alignment
 62118 with the ISO/IEC 9899:1999 standard.

62119 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/130 is applied, restoring wording in the
 62120 RETURN VALUE section to that in the original base document ("An implementation should
 62121 only check for this error if no signal is pending in *set* and it is necessary to wait").

62122 **Issue 7**

62123 The *sigtimedwait()* and *sigwaitinfo()* functions are moved from the Realtime Signals Extension
 62124 option to the Base.

NAME

sigwait — wait for queued signals

SYNOPSIS

```
#include <signal.h>

int sigwait(const sigset_t *restrict set, int *restrict sig);
```

DESCRIPTION

The *sigwait()* function shall select a pending signal from *set*, atomically clear it from the system's set of pending signals, and return that signal number in the location referenced by *sig*. If prior to the call to *sigwait()* there are multiple pending instances of a single signal number, it is implementation-defined whether upon successful return there are any remaining pending signals for that signal number. If the implementation supports queued signals and there are multiple signals queued for the signal number selected, the first such queued signal shall cause a return from *sigwait()* and the remainder shall remain queued. If no signal in *set* is pending at the time of the call, the thread shall be suspended until one or more becomes pending. The signals defined by *set* shall have been blocked at the time of the call to *sigwait()*; otherwise, the behavior is undefined. The effect of *sigwait()* on the signal actions for the signals in *set* is unspecified.

If more than one thread is using *sigwait()* to wait for the same signal, no more than one of these threads shall return from *sigwait()* with the signal number. If more than a single thread is blocked in *sigwait()* for a signal when that signal is generated for the process, it is unspecified which of the waiting threads returns from *sigwait()*. If the signal is generated for a specific thread, as by *pthread_kill()*, only that thread shall return.

Should any of the multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified.

RETURN VALUE

Upon successful completion, *sigwait()* shall store the signal number of the received signal at the location referenced by *sig* and return zero. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *sigwait()* function may fail if:

[EINVAL] The *set* argument contains an invalid or unsupported signal number.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

To provide a convenient way for a thread to wait for a signal, this volume of POSIX.1-2008 provides the *sigwait()* function. For most cases where a thread has to wait for a signal, the *sigwait()* function should be quite convenient, efficient, and adequate.

However, requests were made for a lower-level primitive than *sigwait()* and for semaphores that could be used by threads. After some consideration, threads were allowed to use semaphores and *sem_post()* was defined to be async-signal and async-cancel-safe.

In summary, when it is necessary for code run in response to an asynchronous signal to notify a thread, *sigwait()* should be used to handle the signal. Alternatively, if the implementation provides semaphores, they also can be used, either following *sigwait()* or from within a signal

sigwait()

System Interfaces

62170 handling routine previously registered with *sigaction()*.

62171 FUTURE DIRECTIONS

62172 None.

62173 SEE ALSO

62174 [Section 2.4](#) (on page 484), [Section 2.8.1](#) (on page 497), [pause\(\)](#), [pthread_sigmask\(\)](#), [sigaction\(\)](#),
62175 [sigpending\(\)](#), [sigsuspend\(\)](#), [sigtimedwait\(\)](#)

62176 XBD [<signal.h>](#), [<time.h>](#)

62177 CHANGE HISTORY

62178 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
62179 POSIX Threads Extension.

62180 Issue 6

62181 The **restrict** keyword is added to the *sigwait()* prototype for alignment with the
62182 ISO/IEC 9899:1999 standard.

62183 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/131 is applied, updating the
62184 DESCRIPTION to state that if more than a single thread is blocked in *sigwait()*, it is unspecified
62185 which of the waiting threads returns, and that if a signal is generated for a specific thread only
62186 that thread shall return.

62187 Issue 7

62188 Functionality relating to the Realtime Signals Extension option is moved to the Base.

*System Interfaces***sigwaitinfo()**62189 **NAME**

62190 sigwaitinfo — wait for queued signals

62191 **SYNOPSIS**62192 `#include <signal.h>`62193 `int sigwaitinfo(const sigset_t *restrict set, siginfo_t *restrict info);`62194 **DESCRIPTION**62195 Refer to *sigtimedwait()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

sin()*System Interfaces*62196 **NAME**62197 `sin, sinf, sinl` — sine function62198 **SYNOPSIS**62199 `#include <math.h>`62200 `double sin(double x);`62201 `float sinf(float x);`62202 `long double sinl(long double x);`62203 **DESCRIPTION**

62204 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 62205 conflict between the requirements described here and the ISO C standard is unintentional. This
 62206 volume of POSIX.1-2008 defers to the ISO C standard.

62207 These functions shall compute the sine of their argument x , measured in radians.

62208 An application wishing to check for error situations should set *errno* to zero and call
 62209 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 62210 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 62211 zero, an error has occurred.

62212 **RETURN VALUE**

62213 Upon successful completion, these functions shall return the sine of x .

62214 MX If x is NaN, a NaN shall be returned.

62215 If x is ± 0 , x shall be returned.

62216 If x is subnormal, a range error may occur and x should be returned.

62217 If x is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 62218 defined value shall be returned.

62219 **ERRORS**

62220 These functions shall fail if:

62221 MX **Domain Error** The x argument is $\pm \text{Inf}$.

62222 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 62223 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 62224 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 62225 shall be raised.

62226 These functions may fail if:

62227 MX **Range Error** The value of x is subnormal

62228 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 62229 then *errno* shall be set to [ERANGE]. If the integer expression
 62230 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 62231 floating-point exception shall be raised.

62232 **EXAMPLES**62233 **Taking the Sine of a 45-Degree Angle**

```

62234 #include <math.h>
62235 ...
62236 double radians = 45.0 * M_PI / 180;
62237 double result;
62238 ...
62239 result = sin(radians);

```

62240 **APPLICATION USAGE**

62241 These functions may lose accuracy when their argument is near a multiple of π or is far from 0.0.

62242 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 62243 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

62244 **RATIONALE**

62245 None.

62246 **FUTURE DIRECTIONS**

62247 None.

62248 **SEE ALSO**

62249 *asin()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

62250 XBD Section 4.19 (on page 116), *<math.h>*

62251 **CHANGE HISTORY**

62252 First released in Issue 1. Derived from Issue 1 of the SVID.

62253 **Issue 5**

62254 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes
 62255 in previous issues.

62256 **Issue 6**

62257 The *sinf()* and *sinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

62258 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 62259 revised to align with the ISO/IEC 9899:1999 standard.

62260 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 62261 marked.

sinh()62262 **NAME**

62263 sinh, sinh, sinhl — hyperbolic sine functions

62264 **SYNOPSIS**

```
62265       #include <math.h>

62266       double sinh(double x);
62267       float  sinhf(float x);
62268       long double sinhl(long double x);
```

62269 **DESCRIPTION**

62270 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 62271 conflict between the requirements described here and the ISO C standard is unintentional. This
 62272 volume of POSIX.1-2008 defers to the ISO C standard.

62273 These functions shall compute the hyperbolic sine of their argument *x*.

62274 An application wishing to check for error situations should set *errno* to zero and call
 62275 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 62276 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 62277 zero, an error has occurred.

62278 **RETURN VALUE**

62279 Upon successful completion, these functions shall return the hyperbolic sine of *x*.

62280 If the result would cause an overflow, a range error shall occur and $\pm\text{HUGE\_VAL}$,
 62281 $\pm\text{HUGE\_VALF}$, and $\pm\text{HUGE\_VALL}$ (with the same sign as *x*) shall be returned as appropriate for
 62282 the type of the function.

62283 **MX** If *x* is NaN, a NaN shall be returned.

62284 If *x* is ± 0 or $\pm\text{Inf}$, *x* shall be returned.

62285 If *x* is subnormal, a range error may occur and *x* should be returned.

62286 **ERRORS**

62287 These functions shall fail if:

62288 Range Error The result would cause an overflow.

62289 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 62290 then *errno* shall be set to [ERANGE]. If the integer expression
 62291 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 62292 floating-point exception shall be raised.

62293 These functions may fail if:

62294 **MX** Range Error The value *x* is subnormal.

62295 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 62296 then *errno* shall be set to [ERANGE]. If the integer expression
 62297 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 62298 floating-point exception shall be raised.

62299 **EXAMPLES**

62300 None.

62301 **APPLICATION USAGE**

62302 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 62303 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

62304 **RATIONALE**

62305 None.

62306 **FUTURE DIRECTIONS**

62307 None.

62308 **SEE ALSO**62309 *asinh()*, *cosh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tanh()*

62310 XBD Section 4.19 (on page 116), <math.h>

62311 **CHANGE HISTORY**

62312 First released in Issue 1. Derived from Issue 1 of the SVID.

62313 **Issue 5**

62314 The DESCRIPTION is updated to indicate how an application should check for an error. This
 62315 text was previously published in the APPLICATION USAGE section.

62316 **Issue 6**62317 The *sinhf()* and *sinhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

62318 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 62319 revised to align with the ISO/IEC 9899:1999 standard.

62320 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 62321 marked.

sinl()*System Interfaces*62322 **NAME**

62323 sinl — sine function

62324 **SYNOPSIS**

62325 #include <math.h>

62326 long double sinl(long double x);

62327 **DESCRIPTION**62328 Refer to *sin()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

sleep()**NAME**

sleep — suspend execution for an interval of time

SYNOPSIS

```
#include <unistd.h>
```

```
unsigned sleep(unsigned seconds);
```

DESCRIPTION

The *sleep()* function shall cause the calling thread to be suspended from execution until either the number of realtime seconds specified by the argument *seconds* has elapsed or a signal is delivered to the calling thread and its action is to invoke a signal-catching function or to terminate the process. The suspension time may be longer than requested due to the scheduling of other activity by the system.

If a SIGALRM signal is generated for the calling process during execution of *sleep()* and if the SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *sleep()* returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also unspecified whether it remains pending after *sleep()* returns or it is discarded.

If a SIGALRM signal is generated for the calling process during execution of *sleep()*, except as a result of a prior call to *alarm()*, and if the SIGALRM signal is not being ignored or blocked from delivery, it is unspecified whether that signal has any effect other than causing *sleep()* to return.

If a signal-catching function interrupts *sleep()* and examines or changes either the time a SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or whether the SIGALRM signal is blocked from delivery, the results are unspecified.

If a signal-catching function interrupts *sleep()* and calls *siglongjmp()* or *longjmp()* to restore an environment saved prior to the *sleep()* call, the action associated with the SIGALRM signal and the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also unspecified whether the SIGALRM signal is blocked, unless the signal mask of the process is restored as part of the environment.

XSI Interactions between *sleep()* and *setitimer()* are unspecified.

RETURN VALUE

If *sleep()* returns because the requested time has elapsed, the value returned shall be 0. If *sleep()* returns due to delivery of a signal, the return value shall be the “unslept” amount (the requested time minus the time actually slept) in seconds.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

There are two general approaches to the implementation of the *sleep()* function. One is to use the *alarm()* function to schedule a SIGALRM signal and then suspend the calling thread waiting for that signal. The other is to implement an independent facility. This volume of POSIX.1-2008 permits either approach.

In order to comply with the requirement that no primitive shall change a process attribute unless explicitly described by this volume of POSIX.1-2008, an implementation using SIGALRM must carefully take into account any SIGALRM signal scheduled by previous *alarm()* calls, the action

sleep()*System Interfaces*

previously established for SIGALRM, and whether SIGALRM was blocked. If a SIGALRM has been scheduled before the *sleep()* would ordinarily complete, the *sleep()* must be shortened to that time and a SIGALRM generated (possibly simulated by direct invocation of the signal-catching function) before *sleep()* returns. If a SIGALRM has been scheduled after the *sleep()* would ordinarily complete, it must be rescheduled for the same time before *sleep()* returns. The action and blocking for SIGALRM must be saved and restored.

Historical implementations often implement the SIGALRM-based version using *alarm()* and *pause()*. One such implementation is prone to infinite hangs, as described in *pause()*. Another such implementation uses the C-language *setjmp()* and *longjmp()* functions to avoid that window. That implementation introduces a different problem: when the SIGALRM signal interrupts a signal-catching function installed by the user to catch a different signal, the *longjmp()* aborts that signal-catching function. An implementation based on *sigprocmask()*, *alarm()*, and *sigsuspend()* can avoid these problems.

Despite all reasonable care, there are several very subtle, but detectable and unavoidable, differences between the two types of implementations. These are the cases mentioned in this volume of POSIX.1-2008 where some other activity relating to SIGALRM takes place, and the results are stated to be unspecified. All of these cases are sufficiently unusual as not to be of concern to most applications.

See also the discussion of the term *realtime* in *alarm()*.

Since *sleep()* can be implemented using *alarm()*, the discussion about alarms occurring early under *alarm()* applies to *sleep()* as well.

Application developers should note that the type of the argument *seconds* and the return value of *sleep()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX}, which the ISO C standard sets as 65 535, and any application passing a larger value is restricting its portability. A different type was considered, but historical implementations, including those with a 16-bit **int** type, consistently use either **unsigned** or **int**.

Scheduling delays may cause the process to return from the *sleep()* function significantly after the requested time. In such cases, the return value should be set to zero, since the formula (requested time minus the time actually spent) yields a negative number and *sleep()* returns an **unsigned**.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm(), *getitimer()*, *nanosleep()*, *pause()*, *sigaction()*, *sigsetjmp()*

XBD <unistd.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/132 is applied, making a correction in the RATIONALE section.

*System Interfaces***snprintf()**62417 **NAME**

62418 snprintf — print formatted output

62419 **SYNOPSIS**

62420 #include <stdio.h>

62421 int snprintf(char *restrict *s*, size_t *n*,62422 const char *restrict *format*, ...);62423 **DESCRIPTION**62424 Refer to *fprintf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

socketmark()*System Interfaces***NAME**

socketmark — determine whether a socket is at the out-of-band mark

SYNOPSIS

```
#include <sys/socket.h>

int socketmark(int s);
```

DESCRIPTION

The *socketmark()* function shall determine whether the socket specified by the descriptor *s* is at the out-of-band data mark (see [Section 2.10.12](#), on page 520). If the protocol for the socket supports out-of-band data by marking the stream with an out-of-band data mark, the *socketmark()* function shall return 1 when all data preceding the mark has been read and the out-of-band data mark is the first element in the receive queue. The *socketmark()* function shall not remove the mark from the stream.

RETURN VALUE

Upon successful completion, the *socketmark()* function shall return a value indicating whether the socket is at an out-of-band data mark. If the protocol has marked the data stream and all data preceding the mark has been read, the return value shall be 1; if there is no mark, or if data precedes the mark in the receive queue, the *socketmark()* function shall return 0. Otherwise, it shall return a value of -1 and set *errno* to indicate the error.

ERRORS

The *socketmark()* function shall fail if:

[EBADF]	The <i>s</i> argument is not a valid file descriptor.
[ENOTTY]	The file associated with the <i>s</i> argument is not a socket.

EXAMPLES

None.

APPLICATION USAGE

The use of this function between receive operations allows an application to determine which received data precedes the out-of-band data and which follows the out-of-band data.

There is an inherent race condition in the use of this function. On an empty receive queue, the current read of the location might well be at the “mark”, but the system has no way of knowing that the next data segment that will arrive from the network will carry the mark, and *socketmark()* will return false, and the next read operation will silently consume the mark.

Hence, this function can only be used reliably when the application already knows that the out-of-band data has been seen by the system or that it is known that there is data waiting to be read at the socket (via SIGURG or *select()*). See [Section 2.10.11](#) (on page 520), [Section 2.10.12](#) (on page 520), [Section 2.10.14](#) (on page 521), and *pselect()* for details.

RATIONALE

The *socketmark()* function replaces the historical SIOCATMARK command to *ioctl()* which implemented the same functionality on many implementations. Using a wrapper function follows the adopted conventions to avoid specifying commands to the *ioctl()* function, other than those now included to support XSI STREAMS. The *socketmark()* function could be implemented as follows:

```
#include <sys/ioctl.h>

int socketmark(int s)
{
    int val;
```

System Interfaces

socketmark()

```
62470         if (ioctl(s, SIOCATMARK, &val) == -1)
62471             return(-1);
62472         return(val);
62473     }
```

62474 The use of [ENOTTY] to indicate an incorrect descriptor type matches the historical behavior of
62475 SIOCATMARK.

62476 FUTURE DIRECTIONS

62477 None.

62478 SEE ALSO

62479 [Section 2.10.12](#) (on page 520), *pselect()*, *recv()*, *recvmsg()*

62480 XBD [<sys/socket.h>](#)

62481 CHANGE HISTORY

62482 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

62483 Issue 7

62484 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

socket()*System Interfaces*62485 **NAME**62486 **socket** — create an endpoint for communication62487 **SYNOPSIS**62488 `#include <sys/socket.h>`62489 `int socket(int domain, int type, int protocol);`62490 **DESCRIPTION**62491 The *socket()* function shall create an unbound socket in a communications domain, and return a
62492 file descriptor that can be used in later function calls that operate on sockets.62493 The *socket()* function takes the following arguments:62494 *domain* Specifies the communications domain in which a socket is to be created.62495 *type* Specifies the type of socket to be created.62496 *protocol* Specifies a particular protocol to be used with the socket. Specifying a *protocol*
62497 of 0 causes *socket()* to use an unspecified default protocol appropriate for the
62498 requested socket type.62499 The *domain* argument specifies the address family used in the communications domain. The
62500 address families supported by the system are implementation-defined.62501 Symbolic constants that can be used for the domain argument are defined in the **<sys/socket.h>**
62502 header.62503 The *type* argument specifies the socket type, which determines the semantics of communication
62504 over the socket. The following socket types are defined; implementations may specify additional
62505 socket types:62506 **SOCK_STREAM** Provides sequenced, reliable, bidirectional, connection-mode byte streams,
62507 and may provide a transmission mechanism for out-of-band data.62508 **SOCK_DGRAM** Provides datagrams, which are connectionless-mode, unreliable messages of
62509 fixed maximum length.62510 **SOCK_SEQPACKET**62511 Provides sequenced, reliable, bidirectional, connection-mode transmission
62512 paths for records. A record can be sent using one or more output operations
62513 and received using one or more input operations, but a single operation never
62514 transfers part of more than one record. Record boundaries are visible to the
62515 receiver via the MSG_EOR flag.62516 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address
62517 family. If the *protocol* argument is zero, the default protocol for this address family and type shall
62518 be used. The protocols supported by the system are implementation-defined.62519 The process may need to have appropriate privileges to use the *socket()* function or to create
62520 some sockets.62521 **RETURN VALUE**62522 Upon successful completion, *socket()* shall return a non-negative integer, the socket file
62523 descriptor. Otherwise, a value of -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *socket()* function shall fail if:

[EAFNOSUPPORT]

The implementation does not support the specified address family.

[EMFILE]

All file descriptors available to the process are currently open.

[ENFILE]

No more file descriptors are available for the system.

[EPROTONOSUPPORT]

The protocol is not supported by the address family, or the protocol is not supported by the implementation.

[EPROTOTYPE] The socket type is not supported by the protocol.

The *socket()* function may fail if:

[EACCES]

The process does not have appropriate privileges.

[ENOBUFS]

Insufficient resources were available in the system to perform the operation.

[ENOMEM]

Insufficient memory was available to fulfill the request.

EXAMPLES

None.

APPLICATION USAGE

The documentation for specific address families specifies which protocols each address family supports. The documentation for specific protocols specifies which socket types each protocol supports.

The application can determine whether an address family is supported by trying to create a socket with *domain* set to the protocol in question.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

accept(), *bind()*, *connect()*, *getsockname()*, *getsockopt()*, *listen()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendmsg()*, *setsockopt()*, *shutdown()*, *socketpair()*

XBD [<netinet/in.h>](#), [<sys/socket.h>](#)

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

socketpair()

System Interfaces

62556 **NAME**62557 **socketpair** — create a pair of connected sockets62558 **SYNOPSIS**

```
62559 #include <sys/socket.h>
62560 int socketpair(int domain, int type, int protocol,
62561               int socket_vector[2]);
```

62562 **DESCRIPTION**

62563 The *socketpair()* function shall create an unbound pair of connected sockets in a specified *domain*,
 62564 of a specified *type*, under the protocol optionally specified by the *protocol* argument. The two
 62565 sockets shall be identical. The file descriptors used in referencing the created sockets shall be
 62566 returned in *socket_vector*[0] and *socket_vector*[1].

62567 The *socketpair()* function takes the following arguments:

62568	<i>domain</i>	Specifies the communications domain in which the sockets are to be created.
62569	<i>type</i>	Specifies the type of sockets to be created.
62570	<i>protocol</i>	Specifies a particular protocol to be used with the sockets. Specifying a
62571		<i>protocol</i> of 0 causes <i>socketpair()</i> to use an unspecified default protocol
62572		appropriate for the requested socket type.
62573	<i>socket_vector</i>	Specifies a 2-integer array to hold the file descriptors of the created socket pair.

62574 The *type* argument specifies the socket type, which determines the semantics of communications
 62575 over the socket. The following socket types are defined; implementations may specify additional
 62576 socket types:

62577	SOCK_STREAM	Provides sequenced, reliable, bidirectional, connection-mode byte
62578		streams, and may provide a transmission mechanism for out-of-band
62579		data.
62580	SOCK_DGRAM	Provides datagrams, which are connectionless-mode, unreliable messages
62581		of fixed maximum length.
62582	SOCK_SEQPACKET	Provides sequenced, reliable, bidirectional, connection-mode transmission
62583		paths for records. A record can be sent using one or more output
62584		operations and received using one or more input operations, but a single
62585		operation never transfers part of more than one record. Record
62586		boundaries are visible to the receiver via the MSG_EOR flag.

62587 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address
 62588 family. If the *protocol* argument is zero, the default protocol for this address family and type shall
 62589 be used. The protocols supported by the system are implementation-defined.

62590 The process may need to have appropriate privileges to use the *socketpair()* function or to create
 62591 some sockets.

62592 **RETURN VALUE**

62593 Upon successful completion, this function shall return 0; otherwise, -1 shall be returned and
 62594 *errno* set to indicate the error.

62595 **ERRORS**

62596 The *socketpair()* function shall fail if:

62597	[EAFNOSUPPORT]	
62598		The implementation does not support the specified address family.

System Interfaces

socketpair()

62599	[EMFILE]	All, or all but one, of the file descriptors available to the process are currently open.
62600		
62601	[ENFILE]	No more file descriptors are available for the system.
62602	[EOPNOTSUPP]	The specified protocol does not permit creation of socket pairs.
62603	[EPROTONOSUPPORT]	
62604		The protocol is not supported by the address family, or the protocol is not supported by the implementation.
62605		
62606	[EPROTOTYPE]	The socket type is not supported by the protocol.
62607		The <i>socketpair()</i> function may fail if:
62608	[EACCES]	The process does not have appropriate privileges.
62609	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
62610	[ENOMEM]	Insufficient memory was available to fulfill the request.

62611 **EXAMPLES**

62612 None.

62613 **APPLICATION USAGE**

62614 The documentation for specific address families specifies which protocols each address family supports. The documentation for specific protocols specifies which socket types each protocol supports.

62617 The *socketpair()* function is used primarily with UNIX domain sockets and need not be supported for other domains.

62619 **RATIONALE**

62620 None.

62621 **FUTURE DIRECTIONS**

62622 None.

62623 **SEE ALSO**62624 [socket\(\)](#)62625 XBD [<sys/socket.h>](#)62626 **CHANGE HISTORY**

62627 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

62628 **Issue 7**62629 The description of the [EMFILE] error condition is aligned with the *pipe()* function.

sprintf()*System Interfaces*62630 **NAME**62631 `sprintf` — print formatted output62632 **SYNOPSIS**62633 `#include <stdio.h>`62634 `int sprintf(char *restrict s, const char *restrict format, ...);`62635 **DESCRIPTION**62636 Refer to *fprintf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

sqrt, sqrtf, sqrtl — square root function

SYNOPSIS

```
#include <math.h>

double sqrt(double x);
float sqrtf(float x);
long double sqrtl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall compute the square root of their argument x , $\sqrt{x}$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the square root of x .

For finite values of $x < -0$, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If x is NaN, a NaN shall be returned.

If x is ± 0 or $+\text{Inf}$, x shall be returned.

If x is $-\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

ERRORS

These functions shall fail if:

Domain Error The finite value of x is < -0 , or x is $-\text{Inf}$.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES**Taking the Square Root of 9.0**

```
#include <math.h>
...
double x = 9.0;
double result;
...
result = sqrt(x);
```

sqrt()*System Interfaces*62676 **APPLICATION USAGE**

62677 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
62678 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

62679 **RATIONALE**

62680 None.

62681 **FUTURE DIRECTIONS**

62682 None.

62683 **SEE ALSO**

62684 *feclearexcept()*, *fetestexcept()*, *isnan()*

62685 XBD Section 4.19 (on page 116), **<math.h>**, **<stdio.h>**

62686 **CHANGE HISTORY**

62687 First released in Issue 1. Derived from Issue 1 of the SVID.

62688 **Issue 5**

62689 The DESCRIPTION is updated to indicate how an application should check for an error. This
62690 text was previously published in the APPLICATION USAGE section.

62691 **Issue 6**

62692 The *sqrtof()* and *sqrtrl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

62693 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
62694 revised to align with the ISO/IEC 9899:1999 standard.

62695 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
62696 marked.

*System Interfaces***srand()**62697 **NAME**

62698 srand — pseudo-random number generator

62699 **SYNOPSIS**

62700 #include <stdlib.h>

62701 void srand(unsigned seed);

62702 **DESCRIPTION**62703 Refer to *rand()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

srand48()*System Interfaces*62704 **NAME**

62705 srand48 — seed the uniformly distributed double-precision pseudo-random number generator

62706 **SYNOPSIS**

62707 XSI #include <stdlib.h>

62708 void srand48(long seedval);

62709 **DESCRIPTION**62710 Refer to *drand48()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***srandom()**62711 **NAME**

62712 srandom — seed pseudo-random number generator

62713 **SYNOPSIS**

62714 XSI #include <stdlib.h>

62715 void srandom(unsigned seed);

62716 **DESCRIPTION**62717 Refer to *initstate()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

sscanf()*System Interfaces*62718 **NAME**62719 **sscanf** — convert formatted input62720 **SYNOPSIS**62721 `#include <stdio.h>`62722 `int sscanf(const char *restrict s, const char *restrict format, ...);`62723 **DESCRIPTION**62724 Refer to *fscanf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***stat()**62725 **NAME**

62726 stat — get file status

62727 **SYNOPSIS**

62728 #include <sys/stat.h>

62729 int stat(const char *restrict *path*, struct stat *restrict *buf*);62730 **DESCRIPTION**62731 Refer to *fstatat()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

statvfs()*System Interfaces*62732 **NAME**

62733 statvfs — get file system information

62734 **SYNOPSIS**

62735 #include <sys/statvfs.h>

62736 int statvfs(const char *restrict *path*, struct statvfs *restrict *buf*);62737 **DESCRIPTION**62738 Refer to *fstatvfs()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

stderr, stdin, stdout — standard I/O streams

SYNOPSIS

```
#include <stdio.h>
```

```
extern FILE *stderr, *stdin, *stdout;
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type **FILE**. The *fopen()* function shall create certain descriptive data for a stream and return a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the **<stdio.h>** header and associated with the standard open files.

At program start-up, three streams shall be predefined and need not be opened explicitly: *standard input* (for reading conventional input), *standard output* (for writing conventional output), and *standard error* (for writing diagnostic output). When opened, the standard error stream is not fully buffered; the standard input and standard output streams are fully buffered if and only if the stream can be determined not to refer to an interactive device.

The following symbolic values in **<unistd.h>** define the file descriptors that shall be associated with the C-language *stdin*, *stdout*, and *stderr* when the application is started:

STDIN_FILENO Standard input value, *stdin*. Its value is 0.

STDOUT_FILENO Standard output value, *stdout*. Its value is 1.

STDERR_FILENO Standard error value, *stderr*. Its value is 2.

The *stderr* stream is expected to be open for reading and writing.

RETURN VALUE

None.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fclose(), *feof()*, *ferror()*, *fileno()*, *fopen()*, *fprintf()*, *fread()*, *fscanf()*, *fseek()*, *getc()*, *gets()*, *popen()*, *putc()*, *puts()*, *read()*, *setbuf()*, *setvbuf()*, *tmpfile()*, *ungetc()*, *vfprintf()*

XBD **<stdio.h>**, **<unistd.h>**

62779 **CHANGE HISTORY**

62780 First released in Issue 1.

62781 **Issue 6**

62782 Extensions beyond the ISO C standard are marked.

62783 A note that *stderr* is expected to be open for reading and writing is added to the DESCRIPTION.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***stpcpy()**62784 **NAME**

62785 stpcpy — copy a string and return a pointer to the end of the result

62786 **SYNOPSIS**62787 CX `#include <string.h>`62788 `char *stpcpy(char *restrict s1, const char *restrict s2);`62789 **DESCRIPTION**62790 Refer to *strcpy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

stpncpy()*System Interfaces*62791 **NAME**

62792 stpncpy — copy fixed length string, returning a pointer to the array end

62793 **SYNOPSIS**62794 CX `#include <string.h>`62795 `char *stpncpy(char *restrict s1, const char *restrict s2, size_t size);`62796 **DESCRIPTION**62797 Refer to *strncpy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

strcasemp, strcasemp_l, strncasemp, strncasemp_l — case-insensitive string comparisons

SYNOPSIS

```
#include <strings.h>

int strcasemp(const char *s1, const char *s2);
int strcasemp_l(const char *s1, const char *s2,
    locale_t locale);
int strncasemp(const char *s1, const char *s2, size_t n);
int strncasemp_l(const char *s1, const char *s2,
    size_t n, locale_t locale);
```

DESCRIPTION

The *strcasemp()* and *strcasemp_l()* functions shall compare, while ignoring differences in case, the string pointed to by *s1* to the string pointed to by *s2*. The *strncasemp()* and *strncasemp_l()* functions shall compare, while ignoring differences in case, not more than *n* bytes from the string pointed to by *s1* to the string pointed to by *s2*.

The *strcasemp()* and *strncasemp()* functions use the current locale of the process to determine the case of the characters.

The *strcasemp_l()* and *strncasemp_l()* functions use the locale represented by *locale* to determine the case of the characters.

When the *LC_CTYPE* category of the current locale is from the POSIX locale, *strcasemp()* and *strncasemp()* shall behave as if the strings had been converted to lowercase and then a byte comparison performed. Otherwise, the results are unspecified.

RETURN VALUE

Upon completion, *strcasemp()* and *strcasemp_l()* shall return an integer greater than, equal to, or less than 0, if the string pointed to by *s1* is, ignoring case, greater than, equal to, or less than the string pointed to by *s2*, respectively.

Upon successful completion, *strncasemp()* and *strncasemp_l()* shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by *s1* is, ignoring case, greater than, equal to, or less than the possibly null-terminated array pointed to by *s2*, respectively.

ERRORS

The *strcasemp_l()* and *strncasemp_l()* functions may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

strcasecmp()

System Interfaces

62839 **SEE ALSO**62840 [*wcscasecmp\(\)*](#)62841 XBD [**<strings.h>**](#)62842 **CHANGE HISTORY**

62843 First released in Issue 4, Version 2.

62844 **Issue 5**

62845 Moved from X/OPEN UNIX extension to BASE.

62846 **Issue 7**62847 The *strcasecmp()* and *strncasecmp()* functions are moved from the XSI option to the Base.62848 The *strcasecmp_l()* and *strncasecmp_l()* functions are added from The Open Group Technical
62849 Standard, 2006, Extended API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

62850 **NAME**

62851 strcat — concatenate two strings

62852 **SYNOPSIS**

62853 #include <string.h>

62854 char *strcat(char *restrict s1, const char *restrict s2);

62855 **DESCRIPTION**

62856 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 62857 conflict between the requirements described here and the ISO C standard is unintentional. This
 62858 volume of POSIX.1-2008 defers to the ISO C standard.

62859 The *strcat()* function shall append a copy of the string pointed to by *s2* (including the
 62860 terminating NUL character) to the end of the string pointed to by *s1*. The initial byte of *s2*
 62861 overwrites the NUL character at the end of *s1*. If copying takes place between objects that
 62862 overlap, the behavior is undefined.

62863 **RETURN VALUE**62864 The *strcat()* function shall return *s1*; no return value is reserved to indicate an error.62865 **ERRORS**

62866 No errors are defined.

62867 **EXAMPLES**

62868 None.

62869 **APPLICATION USAGE**

62870 This version is aligned with the ISO C standard; this does not affect compatibility with XPG3
 62871 applications. Reliable error detection by this function was never guaranteed.

62872 **RATIONALE**

62873 None.

62874 **FUTURE DIRECTIONS**

62875 None.

62876 **SEE ALSO**62877 [strncat\(\)](#)62878 XBD [<string.h>](#)62879 **CHANGE HISTORY**

62880 First released in Issue 1. Derived from Issue 1 of the SVID.

62881 **Issue 6**62882 The *strcat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

strchr()*System Interfaces*62883 **NAME**

62884 strchr — string scanning operation

62885 **SYNOPSIS**

62886 #include <string.h>

62887 char *strchr(const char *s, int c);

62888 **DESCRIPTION**

62889 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 62890 conflict between the requirements described here and the ISO C standard is unintentional. This
 62891 volume of POSIX.1-2008 defers to the ISO C standard.

62892 The *strchr()* function shall locate the first occurrence of *c* (converted to a **char**) in the string
 62893 pointed to by *s*. The terminating NUL character is considered to be part of the string.

62894 **RETURN VALUE**

62895 Upon completion, *strchr()* shall return a pointer to the byte, or a null pointer if the byte was not
 62896 found.

62897 **ERRORS**

62898 No errors are defined.

62899 **EXAMPLES**

62900 None.

62901 **APPLICATION USAGE**

62902 None.

62903 **RATIONALE**

62904 None.

62905 **FUTURE DIRECTIONS**

62906 None.

62907 **SEE ALSO**62908 [strrchr\(\)](#)62909 XBD [<string.h>](#)62910 **CHANGE HISTORY**

62911 First released in Issue 1. Derived from Issue 1 of the SVID.

62912 **Issue 6**

62913 Extensions beyond the ISO C standard are marked.

NAME

strcmp — compare two strings

SYNOPSIS

```
#include <string.h>

int strcmp(const char *s1, const char *s2);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *strcmp()* function shall compare the string pointed to by *s1* to the string pointed to by *s2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings being compared.

RETURN VALUE

Upon completion, *strcmp()* shall return an integer greater than, equal to, or less than 0, if the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*, respectively.

ERRORS

No errors are defined.

EXAMPLES**Checking a Password Entry**

The following example compares the information read from standard input to the value of the name of the user entry. If the *strcmp()* function returns 0 (indicating a match), a further check will be made to see if the user entered the proper old password. The *crypt()* function shall encrypt the old password entered by the user, using the value of the encrypted password in the **passwd** structure as the salt. If this value matches the value of the encrypted **passwd** in the structure, the entered password *oldpasswd* is the correct user's password. Finally, the program encrypts the new password so that it can store the information in the **passwd** structure.

```
#include <string.h>
#include <unistd.h>
#include <stdio.h>
...
int valid_change;
struct passwd *p;
char user[100];
char oldpasswd[100];
char newpasswd[100];
char savepasswd[100];
...
if (strcmp(p->pw_name, user) == 0) {
    if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
        strcpy(savepasswd, crypt(newpasswd, user));
        p->pw_passwd = savepasswd;
        valid_change = 1;
    }
    else {
```

strcmp()*System Interfaces*

```
62960         fprintf(stderr, "Old password is not valid\n");
62961     }
62962 }
62963 ...
```

APPLICATION USAGE

62965 None.

RATIONALE

62967 None.

FUTURE DIRECTIONS

62969 None.

SEE ALSO

62971 [*strncmp\(\)*](#)

62972 XBD [*<string.h>*](#)

CHANGE HISTORY

62974 First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

62976 Extensions beyond the ISO C standard are marked.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

strcoll, strcoll_l — string comparison using collating information

SYNOPSIS

```
#include <string.h>

int strcoll(const char *s1, const char *s2);
int strcoll_l(const char *s1, const char *s2,
               locale_t locale);
```

DESCRIPTION

For *strcoll()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *strcoll()* and *strcoll_l()* functions shall compare the string pointed to by *s1* to the string pointed to by *s2*, both interpreted as appropriate to the *LC_COLLATE* category of the current locale, or of the locale represented by *locale*, respectively.

The *strcoll()* and *strcoll_l()* functions shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strcoll()*, or *strcoll_l()* then check *errno*.

RETURN VALUE

Upon successful completion, *strcoll()* shall return an integer greater than, equal to, or less than 0, according to whether the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2* when both are interpreted as appropriate to the current locale. On error, *strcoll()* may set *errno*, but no return value is reserved to indicate an error.

Upon successful completion, *strcoll_l()* shall return an integer greater than, equal to, or less than 0, according to whether the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2* when both are interpreted as appropriate to the locale represented by *locale*. On error, *strcoll_l()* may set *errno*, but no return value is reserved to indicate an error.

ERRORS

These functions may fail if:

[EINVAL] The *s1* or *s2* arguments contain characters outside the domain of the collating sequence.

The *strcoll_l()* function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES**Comparing Nodes**

The following example uses an application-defined function, *node_compare()*, to compare two nodes based on an alphabetical ordering of the *string* field.

```
#include <string.h>
...
struct node { /* These are stored in the table. */
    char *string;
    int length;
};
...
```

strcoll()*System Interfaces*

```

63020     int node_compare(const void *node1, const void *node2)
63021     {
63022         return strcoll(((const struct node *)node1)->string,
63023             ((const struct node *)node2)->string);
63024     }
63025     ...

```

63026 APPLICATION USAGE

63027 The *strxfrm()* and *strcmp()* functions should be used for sorting large lists.

63028 RATIONALE

63029 None.

63030 FUTURE DIRECTIONS

63031 None.

63032 SEE ALSO

63033 *alphasort()*, *strcmp()*, *strxfrm()*

63034 XBD <string.h>

63035 CHANGE HISTORY

63036 First released in Issue 3.

63037 Issue 5

63038 The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.

63039 Issue 6

63040 Extensions beyond the ISO C standard are marked.

63041 The following new requirements on POSIX implementations derive from alignment with the
 63042 Single UNIX Specification:

- 63043 • The [EINVAL] optional error condition is added.

63044 An example is added.

63045 Issue 7

63046 The *strcoll_l()* function is added from The Open Group Technical Standard, 2006, Extended API
 63047 Set Part 4.

NAME

strcpy, strcpy — copy a string and return a pointer to the end of the result

SYNOPSIS

```
#include <string.h>
```

```
CX char *strcpy(char *restrict s1, const char *restrict s2);
char *strcpy(char *restrict s1, const char *restrict s2);
```

DESCRIPTION

CX For *strcpy()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

CX The *strcpy()* and *strcpy()* functions shall copy the string pointed to by *s2* (including the terminating NUL character) into the array pointed to by *s1*.

If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

CX The *strcpy()* function shall return a pointer to the terminating NUL character copied into the *s1* buffer.

The *strcpy()* function shall return *s1*.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES**Construction of a Multi-Part Message in a Single Buffer**

```
#include <string.h>
#include <stdio.h>

int
main (void)
{
    char buffer[10];
    char *name = buffer;

    name = strcpy (strcpy (strcpy (name, "ice"), "-"), "cream");
    puts (buffer);
    return 0;
}
```

Initializing a String

The following example copies the string "-----" into the *permstring* variable.

```
#include <string.h>
...
static char permstring[11];
...
strcpy (permstring, "-----");
...
```

strcpy()*System Interfaces***Storing a Key and Data**

The following example allocates space for a key using *malloc()* then uses *strcpy()* to place the key there. Then it allocates space for data using *malloc()*, and uses *strcpy()* to place data there. (The user-defined function *dbfree()* frees memory previously allocated to an array of type **struct element** *.)

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
...
/* Structure used to read data and store it. */
struct element {
    char *key;
    char *data;
};

struct element *tbl, *curtbl;
char *key, *data;
int count;
...
void dbfree(struct element *, int);
...
if ((curtbl->key = malloc(strlen(key) + 1)) == NULL) {
    perror("malloc"); dbfree(tbl, count); return NULL;
}
strcpy(curtbl->key, key);

if ((curtbl->data = malloc(strlen(data) + 1)) == NULL) {
    perror("malloc"); free(curtbl->key); dbfree(tbl, count); return NULL;
}
strcpy(curtbl->data, data);
...
```

APPLICATION USAGE

Character movement is performed differently in different implementations. Thus, overlapping moves may yield surprises.

This version is aligned with the ISO C standard; this does not affect compatibility with XPG3 applications. Reliable error detection by this function was never guaranteed.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncpy(), *wcscpy()*

XBD **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

63132 **Issue 6**

63133 The *strcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

63134 **Issue 7**

63135 The *strcpy()* function is added from The Open Group Technical Standard, 2006, Extended API
63136 Set Part 1.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

strcspn()*System Interfaces*63137 **NAME**

63138 strcspn — get the length of a complementary substring

63139 **SYNOPSIS**

63140 #include <string.h>

63141 size_t strcspn(const char *s1, const char *s2);

63142 **DESCRIPTION**

63143 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 63144 conflict between the requirements described here and the ISO C standard is unintentional. This
 63145 volume of POSIX.1-2008 defers to the ISO C standard.

63146 The *strcspn()* function shall compute the length (in bytes) of the maximum initial segment of the
 63147 string pointed to by *s1* which consists entirely of bytes *not* from the string pointed to by *s2*.

63148 **RETURN VALUE**

63149 The *strcspn()* function shall return the length of the computed segment of the string pointed to
 63150 by *s1*; no return value is reserved to indicate an error.

63151 **ERRORS**

63152 No errors are defined.

63153 **EXAMPLES**

63154 None.

63155 **APPLICATION USAGE**

63156 None.

63157 **RATIONALE**

63158 None.

63159 **FUTURE DIRECTIONS**

63160 None.

63161 **SEE ALSO**63162 *strspn()*

63163 XBD <string.h>

63164 **CHANGE HISTORY**

63165 First released in Issue 1. Derived from Issue 1 of the SVID.

63166 **Issue 5**

63167 The RETURN VALUE section is updated to indicate that *strcspn()* returns the length of *s1*, and
 63168 not *s1* itself as was previously stated.

63169 **Issue 6**

63170 The Open Group Corrigendum U030/1 is applied. The text of the RETURN VALUE section is
 63171 updated to indicate that the computed segment length is returned, not the *s1* length.

NAME

strdup, strndup — duplicate a specific number of bytes from a string

SYNOPSIS

```
#include <string.h>

char *strdup(const char *s);
char *strndup(const char *s, size_t size);
```

DESCRIPTION

The *strdup()* function shall return a pointer to a new string, which is a duplicate of the string pointed to by *s*. The returned pointer can be passed to *free()*. A null pointer is returned if the new string cannot be created.

The *strndup()* function shall be equivalent to the *strdup()* function, duplicating the provided *s* in a new block of memory allocated as if by using *malloc()*, with the exception being that *strndup()* copies at most *size* plus one bytes into the newly allocated memory, terminating the new string with a NUL character. If the length of *s* is larger than *size*, only *size* bytes shall be duplicated. If *size* is larger than the length of *s*, all bytes in *s* shall be copied into the new memory buffer, including the terminating NUL character. The newly created string shall always be properly terminated.

RETURN VALUE

The *strdup()* function shall return a pointer to a new string on success. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

Upon successful completion, the *strndup()* function shall return a pointer to the newly allocated memory containing the duplicated string. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

[ENOMEM] Storage space available is insufficient.

EXAMPLES

None.

APPLICATION USAGE

For functions that allocate memory as if by *malloc()*, the application should release such memory when it is no longer required by a call to *free()*. For *strdup()* and *strndup()*, this is the return value.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

free(), *wcsdup()*

XBD *<string.h>*

CHANGE HISTORY

First released in Issue 4, Version 2.

strdup()*System Interfaces*63213 **Issue 5**

63214 Moved from X/OPEN UNIX extension to BASE.

63215 **Issue 7**63216 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOMEM]
63217 error to become a “shall fail” error.63218 The *strdup()* function is moved from the XSI option to the Base.63219 The *strndup()* function is added from The Open Group Technical Standard, 2006, Extended API
63220 Set Part 1.63221 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by
63222 *malloc()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

strerror()**NAME**

strerror, strerror_l, strerror_r — get error message string

SYNOPSIS

```
#include <string.h>

char *strerror(int errnum);
char *strerror_l(int errnum, locale_t locale);
int strerror_r(int errnum, char *strerrbuf, size_t buflen);
```

DESCRIPTION

For *strerror()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *strerror()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return a pointer to it. Typically, the values for *errnum* come from *errno*, but *strerror()* shall map any value of type **int** to a message.

The string pointed to shall not be modified by the application. The string may be overwritten by a subsequent call to *strerror()*.

The string may be overwritten by a subsequent call to *strerror_l()* in the same thread.

The contents of the error message strings returned by *strerror()* should be determined by the setting of the *LC_MESSAGES* category in the current locale.

The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls *strerror()*.

The *strerror()* and *strerror_l()* functions shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strerror()*, then check *errno*.

The *strerror()* function need not be thread-safe.

The *strerror_l()* function shall map the error number in *errnum* to a locale-dependent error message string in the locale represented by *locale* and shall return a pointer to it.

The *strerror_r()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.

If the value of *errnum* is a valid error number, the message string shall indicate what error occurred; otherwise, if these functions complete successfully, the message string shall indicate that an unknown error occurred.

RETURN VALUE

Upon completion, whether successful or not, *strerror()* shall return a pointer to the generated message string. On error *errno* may be set, but no return value is reserved to indicate an error.

Upon successful completion, *strerror_l()* shall return a pointer to the generated message string. If *errnum* is not a valid error number, *errno* may be set to [EINVAL], but a pointer to a message string shall still be returned. If any other error occurs, *errno* shall be set to indicate the error and a null pointer shall be returned.

Upon successful completion, *strerror_r()* shall return 0. Otherwise, an error number shall be returned to indicate the error.

strerror()*System Interfaces***ERRORS**

63265
63266 These functions may fail if:

63267 CX [EINVAL] The value of *errnum* is not a valid error number.

63268 The *strerror_l()* function may fail if:

63269 CX [EINVAL] The *locale* argument is not a valid locale object handle.

63270 The *strerror_r()* function may fail if:

63271 CX [ERANGE] Insufficient storage was supplied via *strrdbuf* and *buflen* to contain the
63272 generated message string.

EXAMPLES

63273
63274 None.

APPLICATION USAGE

63276 Historically in some implementations, calls to *perror()* would overwrite the string that the
63277 pointer returned by *strerror()* points to. Such implementations did not conform to the ISO C
63278 standard; however, application developers should be aware of this behavior if they wish their
63279 applications to be portable to such implementations.

RATIONALE

63281 The *strerror_l()* function is required to be thread-safe, thereby eliminating the need for an
63282 equivalent to the *strerror_r()* function.

63283 Earlier versions of this standard did not explicitly require that the error message strings returned
63284 by *strerror()* and *strerror_r()* provide any information about the error. This version of the
63285 standard requires a meaningful message for any successful completion.

63286 Since no return value is reserved to indicate a *strerror()* error, but all calls (whether successful or
63287 not) must return a pointer to a message string, on error *strerror()* can return a pointer to an
63288 empty string or a pointer to a meaningful string that can be printed.

63289 Note that the [EINVAL] error condition is a may fail error. If an invalid error number is supplied
63290 as the value of *errnum*, applications should be prepared to handle any of the following:

- 63291 1. Error (with no meaningful message): *errno* is set to [EINVAL], the return value is a pointer
63292 to an empty string.
- 63293 2. Successful completion: *errno* is unchanged and the return value points to a string like
63294 "unknown error" or "error number xxx" (where *xxx* is the value of *errnum*).
- 63295 3. Combination of #1 and #2: *errno* is set to [EINVAL] and the return value points to a string
63296 like "unknown error" or "error number xxx" (where *xxx* is the value of *errnum*).
63297 Since applications frequently use the return value of *strerror()* as an argument to
63298 functions like *fprintf()* (without checking the return value) and since applications have no
63299 way to parse an error message string to determine whether *errnum* represents a valid
63300 error number, implementations are encouraged to implement #3. Similarly,
63301 implementations are encouraged to have *strerror_r()* return [EINVAL] and put a string
63302 like "unknown error" or "error number xxx" in the buffer pointed to by *strrdbuf*
63303 when the value of *errnum* is not a valid error number.

FUTURE DIRECTIONS

63304
63305 None.

SEE ALSO*perror()*

XBD <string.h>

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

A note indicating that the *strerror()* function need not be reentrant is added to the DESCRIPTION.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the fact that *errno* may be set is added.
- The [EINVAL] optional error condition is added.

The normative text is updated to avoid use of the term “must” for application requirements.

The *strerror_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.

The *strerror_r()* function is marked as part of the Thread-Safe Functions option.

Issue 7

Austin Group Interpretation 1003.1-2001 #072 is applied, updating the ERRORS section.

Austin Group Interpretation 1003.1-2001 #156 is applied.

Austin Group Interpretation 1003.1-2001 #187 is applied, clarifying the behavior when the generated error message is an empty string.

SD5-XSH-ERN-191 is applied, updating the APPLICATION USAGE section.

The *strerror_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

The *strerror_r()* function is moved from the Thread-Safe Functions option to the Base.

strfmon()*System Interfaces*63333 **NAME**63334 **strfmon, strfmon_l** — convert monetary value to a string63335 **SYNOPSIS**

```

63336 #include <monetary.h>
63337
63337 ssize_t strfmon(char *restrict s, size_t maxsize,
63338               const char *restrict format, ...);
63339
63339 ssize_t strfmon_l(char *restrict s, size_t maxsize,
63340                 locale_t locale, const char *restrict format, ...);

```

63341 **DESCRIPTION**

63342 The *strfmon()* function shall place characters into the array pointed to by *s* as controlled by the
 63343 string pointed to by *format*. No more than *maxsize* bytes are placed into the array.

63344 The format is a character string, beginning and ending in its initial state, if any, that contains two
 63345 types of objects: *plain characters*, which are simply copied to the output stream, and *conversion*
 63346 *specifications*, each of which shall result in the fetching of zero or more arguments which are
 63347 converted and formatted. The results are undefined if there are insufficient arguments for the
 63348 format. If the format is exhausted while arguments remain, the excess arguments are simply
 63349 ignored.

63350 The application shall ensure that a conversion specification consists of the following sequence:

- 63351 • A '%' character
- 63352 • Optional flags
- 63353 • Optional field width
- 63354 • Optional left precision
- 63355 • Optional right precision
- 63356 • A required conversion specifier character that determines the conversion to be performed

63357 The *strfmon_l()* function shall be equivalent to the *strfmon()* function, except that the locale data
 63358 used is from the locale represented by *locale*.

63359 **Flags**

63360 One or more of the following optional flags can be specified to control the conversion:

- 63361 =*f* An '=' followed by a single character *f* which is used as the numeric fill character. In
 63362 order to work with precision or width counts, the fill character shall be a single byte
 63363 character; if not, the behavior is undefined. The default numeric fill character is the
 63364 <space>. This flag does not affect field width filling which always uses the <space>.
 63365 This flag is ignored unless a left precision (see below) is specified.
- 63366 Do not format the currency amount with grouping characters. The default is to insert
 63367 the grouping characters if defined for the current locale.
- 63368 + or (Specify the style of representing positive and negative currency amounts. Only one of
 63369 '+' or '(' may be specified. If '+' is specified, the locale's equivalent of '+' and '-'
 63370 are used (for example, in many locales, the empty string if positive and '-' if
 63371 negative). If '(' is specified, negative amounts are enclosed within parentheses. If
 63372 neither flag is specified, the '+' style is used.
- 63373 ! Suppress the currency symbol from the output conversion.

63374 – Specify the alignment. If this flag is present the result of the conversion is left-justified
 63375 (padded to the right) rather than right-justified. This flag shall be ignored unless a field
 63376 width (see below) is specified.

63377 Field Width

63378 *w* A decimal digit string *w* specifying a minimum field width in bytes in which the result
 63379 of the conversion is right-justified (or left-justified if the flag '*-*' is specified). The
 63380 default is 0.

63381 Left Precision

63382 *#n* A '*#*' followed by a decimal digit string *n* specifying a maximum number of digits
 63383 expected to be formatted to the left of the radix character. This option can be used to
 63384 keep the formatted output from multiple calls to the *strfmon()* function aligned in the
 63385 same columns. It can also be used to fill unused positions with a special character as in
 63386 "\$***123.45". This option causes an amount to be formatted as if it has the number
 63387 of digits specified by *n*. If more than *n* digit positions are required, this conversion
 63388 specification is ignored. Digit positions in excess of those actually required are filled
 63389 with the numeric fill character (see the *=f* flag above).

63390 If grouping has not been suppressed with the '*s*' flag, and it is defined for the current
 63391 locale, grouping separators are inserted before the fill characters (if any) are added.
 63392 Grouping separators are not applied to fill characters even if the fill character is a digit.

63393 To ensure alignment, any characters appearing before or after the number in the
 63394 formatted output such as currency or sign symbols are padded as necessary with
 63395 <space> characters to make their positive and negative formats an equal length.

63396 Right Precision

63397 *.p* A <period> followed by a decimal digit string *p* specifying the number of digits after
 63398 the radix character. If the value of the right precision *p* is 0, no radix character appears.
 63399 If a right precision is not included, a default specified by the current locale is used. The
 63400 amount being formatted is rounded to the specified number of digits prior to
 63401 formatting.

63402 Conversion Specifier Characters

63403 The conversion specifier characters and their meanings are:

63404 *i* The **double** argument is formatted according to the locale's international currency
 63405 format (for example, in the US: USD 1,234.56). If the argument is $\pm\text{Inf}$ or NaN, the result
 63406 of the conversion is unspecified.

63407 *n* The **double** argument is formatted according to the locale's national currency format
 63408 (for example, in the US: \$1,234.56). If the argument is $\pm\text{Inf}$ or NaN, the result of the
 63409 conversion is unspecified.

63410 *%* Convert to a '*%*'; no argument is converted. The entire conversion specification shall
 63411 be *%%*.

strfmon()*System Interfaces***Locale Information**

The *LC_MONETARY* category of the locale of the process affects the behavior of this function including the monetary radix character (which may be different from the numeric radix character affected by the *LC_NUMERIC* category), the grouping separator, the currency symbols, and formats. The international currency symbol should be conformant with the ISO 4217:2001 standard.

If the value of *maxsize* is greater than {SSIZE_MAX}, the result is implementation-defined.

RETURN VALUE

If the total number of resulting bytes including the terminating null byte is not more than *maxsize*, these functions shall return the number of bytes placed into the array pointed to by *s*, not including the terminating NUL character. Otherwise, -1 shall be returned, the contents of the array are unspecified, and *errno* shall be set to indicate the error.

ERRORS

These functions shall fail if:

[E2BIG] Conversion stopped due to lack of space in the buffer.

The *strfmon_l()* function may fail if:

[EINVAL] *locale* is not a valid locale object.

EXAMPLES

Given a locale for the US and the values 123.45, -123.45, and 3456.781, the following output might be produced. Square brackets (" [] ") are used in this example to delimit the output.

%n	[\$123.45]	Default formatting
	[-\$123.45]	
	[\$3,456.78]	
%11n	[\$123.45]	Right align within an 11-character field
	[-\$123.45]	
	[\$3,456.78]	
%#5n	[\$ 123.45]	Aligned columns for values up to 99 999
	[-\$ 123.45]	
	[\$ 3,456.78]	
%=*#5n	[\$***123.45]	Specify a fill character
	[-\$***123.45]	
	[\$*3,456.78]	
%=0#5n	[\$000123.45]	Fill characters do not use grouping
	[-\$000123.45]	even if the fill character is a digit
	[\$03,456.78]	
%^#5n	[\$ 123.45]	Disable the grouping separator
	[-\$ 123.45]	
	[\$ 3456.78]	
%^#5.0n	[\$ 123]	Round off to whole units
	[-\$ 123]	
	[\$ 3457]	
%^#5.4n	[\$ 123.4500]	Increase the precision
	[-\$ 123.4500]	
	[\$ 3456.7810]	

63456 % (#5n [\$ 123.45] Use an alternative pos/neg style
 63457 [(\$ 123.45)]
 63458 [\$ 3,456.78]
 63459 %! (#5n [123.45] Disable the currency symbol
 63460 [(123.45)]
 63461 [3,456.78]
 63462 %-14#5.4n [\$ 123.4500] Left-justify the output
 63463 [-\$ 123.4500]
 63464 [\$ 3,456.7810]
 63465 %14#5.4n [\$ 123.4500] Corresponding right-justified output
 63466 [-\$ 123.4500]
 63467 [\$ 3,456.7810]

63468 See also the EXAMPLES section in *fprintf()*.

63469 APPLICATION USAGE

63470 None.

63471 RATIONALE

63472 None.

63473 FUTURE DIRECTIONS

63474 Lowercase conversion characters are reserved for future standards use and uppercase for
 63475 implementation-defined use.

63476 SEE ALSO

63477 *fprintf()*, *localeconv()*

63478 XBD <monetary.h>

63479 CHANGE HISTORY

63480 First released in Issue 4.

63481 Issue 5

63482 Moved from ENHANCED118N to BASE.

63483 The [ENOSYS] error is removed.

63484 Text is added to the DESCRIPTION warning about values of *maxsize* that are greater than
 63485 {SSIZE_MAX}.

63486 Issue 6

63487 The normative text is updated to avoid use of the term “must” for application requirements.

63488 The **restrict** keyword is added to the *strfmon()* prototype for alignment with the
 63489 ISO/IEC 9899:1999 standard.

63490 The EXAMPLES section is reworked, clarifying the output format.

63491 Issue 7

63492 SD5-XSH-ERN-29 is applied, updating the examples for % (#5n and %! (#5n.

63493 SD5-XSH-ERN-233 is applied, changing the definition of the '+' or '(' flags to refer to
 63494 multiple locales.

63495 The *strfmon()* function is moved from the XSI option to the Base.

strfmon()*System Interfaces*63496
63497

The *strfmon_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

strftime()**NAME**

strftime, strftime_l — convert date and time to a string

SYNOPSIS

```
#include <time.h>

size_t strftime(char *restrict s, size_t maxsize,
    const char *restrict format, const struct tm *restrict timeptr);
size_t strftime_l(char *restrict s, size_t maxsize,
    const char *restrict format, const struct tm *restrict timeptr,
    locale_t locale);
```

DESCRIPTION

For *strftime()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *strftime()* function shall place bytes into the array pointed to by *s* as controlled by the string pointed to by *format*. The format is a character string, beginning and ending in its initial shift state, if any. The format string consists of zero or more conversion specifications and ordinary characters.

Each conversion specification is introduced by the '%' character after which the following appear in sequence:

- An optional flag:

- 0 The zero character ('0'), which specifies that the character used as the padding character is '0',
- + The <plus-sign> character ('+'), which specifies that the character used as the padding character is '0', and that if and only if the field being produced consumes more than four bytes to represent a year (for %F, %G, or %Y) or more than two bytes to represent the year divided by 100 (for %C) then a leading <plus-sign> character shall be included if the year being processed is greater than or equal to zero or a leading minus-sign character ('-') shall be included if the year is less than zero.

The default padding character is unspecified.

- An optional minimum field width. If the converted value, including any leading '+' or '-' sign, has fewer bytes than the minimum field width and the padding character is not the NUL character, the output shall be padded on the left (after any leading '+' or '-' sign) with the padding character.
- An optional E or O modifier.
- A terminating conversion specifier character that indicates the type of conversion to be applied.

The results are unspecified if more than one flag character is specified, a flag character is specified without a minimum field width; a minimum field width is specified without a flag character; a modifier is specified with a flag or with a minimum field width; or if a minimum field width is specified for any conversion specifier other than C, F, G, or Y.

All ordinary characters (including the terminating NUL character) are copied unchanged into the array. If copying takes place between objects that overlap, the behavior is undefined. No more than *maxsize* bytes are placed into the array. Each conversion specifier is replaced by appropriate characters as described in the following list. The appropriate characters are

strftime()*System Interfaces*

- 63542 determined using the *LC_TIME* category of the current locale and by the values of zero or more
 63543 members of the broken-down time structure pointed to by *timeptr*, as specified in brackets in the
 63544 description. If any of the specified values are outside the normal range, the characters stored are
 63545 unspecified.
- 63546 CX The *strftime_l()* function shall be equivalent to the *strftime()* function, except that the locale data
 63547 used is from the locale represented by *locale*.
- 63548 Local timezone information is used as though *strftime()* called *tzset()*.
- 63549 The following conversion specifiers shall be supported:
- 63550 a Replaced by the locale's abbreviated weekday name. [*tm_wday*]
- 63551 A Replaced by the locale's full weekday name. [*tm_wday*]
- 63552 b Replaced by the locale's abbreviated month name. [*tm_mon*]
- 63553 B Replaced by the locale's full month name. [*tm_mon*]
- 63554 c Replaced by the locale's appropriate date and time representation. (See the Base
 63555 Definitions volume of POSIX.1-2008, <**time.h**>.)
- 63556 C Replaced by the year divided by 100 and truncated to an integer, as a decimal number.
 63557 [*tm_year*]
- 63558 If a minimum field width is not specified, the number of characters placed into the
 63559 array pointed to by *s* will be the number of digits in the year divided by 100 or two,
 63560 CX whichever is greater. If a minimum field width is specified, the number of characters
 63561 placed into the array pointed to by *s* will be the number of digits in the year divided by
 63562 100 or the minimum field width, whichever is greater.
- 63563 d Replaced by the day of the month as a decimal number [01,31]. [*tm_mday*]
- 63564 D Equivalent to %m/%d/%y. [*tm_mon, tm_mday, tm_year*]
- 63565 e Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded
 63566 by a space. [*tm_mday*]
- 63567 CX F Equivalent to %+4Y-%m-%d if no flag and no minimum field width are specified.
 63568 [*tm_year, tm_mon, tm_mday*]
- 63569 CX If a minimum field width of *x* is specified, the year shall be output as if by the *Y*
 63570 specifier (described below) with whatever flag was given and a minimum field width
 63571 of *x*-6. If *x* is less than 6, the behavior shall be as if *x* equalled 6.
- 63572 If the minimum field width is specified to be 10, and the year is four digits long, then
 63573 the output string produced will match the ISO 8601:2004 standard subclause 4.1.2.2
 63574 complete representation, extended format date representation of a specific day. If a +
 63575 flag is specified, a minimum field width of *x* is specified, and *x*-7 bytes are sufficient to
 63576 hold the digits of the year (not including any needed sign character), then the output
 63577 will match the ISO 8601:2004 standard subclause 4.1.2.4 complete representation,
 63578 expanded format date representation of a specific day.
- 63579 g Replaced by the last 2 digits of the week-based year (see below) as a decimal number
 63580 [00,99]. [*tm_year, tm_wday, tm_yday*]
- 63581 G Replaced by the week-based year (see below) as a decimal number (for example, 1977).
 63582 [*tm_year, tm_wday, tm_yday*]

System Interfaces

strftime()

63583	CX	If a minimum field width is specified, the number of characters placed into the array pointed to by <i>s</i> will be the number of digits and leading sign characters (if any) in the year, or the minimum field width, whichever is greater.
63584		
63585		
63586	h	Equivalent to %b. [<i>tm_mon</i>]
63587	H	Replaced by the hour (24-hour clock) as a decimal number [00,23]. [<i>tm_hour</i>]
63588	I	Replaced by the hour (12-hour clock) as a decimal number [01,12]. [<i>tm_hour</i>]
63589	j	Replaced by the day of the year as a decimal number [001,366]. [<i>tm_yday</i>]
63590	m	Replaced by the month as a decimal number [01,12]. [<i>tm_mon</i>]
63591	M	Replaced by the minute as a decimal number [00,59]. [<i>tm_min</i>]
63592	n	Replaced by a <newline>.
63593	p	Replaced by the locale's equivalent of either a.m. or p.m. [<i>tm_hour</i>]
63594	CX	Replaced by the time in a.m. and p.m. notation; in the POSIX locale this shall be equivalent to %I:%M:%S %p. [<i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i>]
63595		
63596	R	Replaced by the time in 24-hour notation (%H:%M). [<i>tm_hour</i> , <i>tm_min</i>]
63597	S	Replaced by the second as a decimal number [00,60]. [<i>tm_sec</i>]
63598	t	Replaced by a <tab>.
63599	T	Replaced by the time (%H:%M:%S). [<i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i>]
63600	u	Replaced by the weekday as a decimal number [1,7], with 1 representing Monday. [<i>tm_wday</i>]
63601		
63602	U	Replaced by the week number of the year as a decimal number [00,53]. The first Sunday of January is the first day of week 1; days in the new year before this are in week 0. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
63603		
63604		
63605	V	Replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. Both January 4th and the first Thursday of January are always in week 1. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
63606		
63607		
63608		
63609		
63610	w	Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday. [<i>tm_wday</i>]
63611		
63612	W	Replaced by the week number of the year as a decimal number [00,53]. The first Monday of January is the first day of week 1; days in the new year before this are in week 0. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
63613		
63614		
63615	x	Replaced by the locale's appropriate date representation. (See the Base Definitions volume of POSIX.1-2008, <time.h>.)
63616		
63617	X	Replaced by the locale's appropriate time representation. (See the Base Definitions volume of POSIX.1-2008, <time.h>.)
63618		
63619	y	Replaced by the last two digits of the year as a decimal number [00,99]. [<i>tm_year</i>]
63620	Y	Replaced by the year as a decimal number (for example, 1997). [<i>tm_year</i>]
63621	CX	If a minimum field width is specified, the number of characters placed into the array pointed to by <i>s</i> will be the number of digits and leading sign characters (if any) in the
63622		

strftime()*System Interfaces*

63623		year, or the minimum field width, whichever is greater.
63624	z	Replaced by the offset from UTC in the ISO 8601:2004 standard format (+hhmm or -hhmm), or by no characters if no timezone is determinable. For example, "-0430" means 4 hours 30 minutes behind UTC (west of Greenwich). If <i>tm_isdst</i> is zero, the standard time offset is used. If <i>tm_isdst</i> is greater than zero, the daylight savings time offset is used. If <i>tm_isdst</i> is negative, no characters are returned. [<i>tm_isdst</i>]
63625	CX	If <i>tm_isdst</i> is zero, the standard time offset is used. If <i>tm_isdst</i> is greater than zero, the daylight savings time offset is used. If <i>tm_isdst</i> is negative, no characters are returned. [<i>tm_isdst</i>]
63626		
63627		
63628		
63629	Z	Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists. [<i>tm_isdst</i>]
63630		
63631	%	Replaced by %.
63632		If a conversion specification does not correspond to any of the above, the behavior is undefined.
63633	CX	If a struct tm broken-down time structure is created by <i>localtime()</i> or <i>localtime_r()</i> , or modified by <i>mktime()</i> , and the value of <i>TZ</i> is subsequently modified, the results of the %Z and %z <i>strftime()</i> conversion specifiers are undefined, when <i>strftime()</i> is called with such a broken-down time structure.
63634		
63635		
63636		
63637		If a struct tm broken-down time structure is created or modified by <i>gmtime()</i> or <i>gmtime_r()</i> , it is unspecified whether the result of the %Z and %z conversion specifiers shall refer to UTC or the current local timezone, when <i>strftime()</i> is called with such a broken-down time structure.
63638		
63639		
63640		Modified Conversion Specifiers
63641		Some conversion specifiers can be modified by the E or O modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale (see ERA in XBD Section 7.3.5, on page 158), the behavior shall be as if the unmodified conversion specification were used.
63642		
63643		
63644		
63645		
63646	%Ec	Replaced by the locale's alternative appropriate date and time representation.
63647	%EC	Replaced by the name of the base year (period) in the locale's alternative representation.
63648		
63649	%Ex	Replaced by the locale's alternative date representation.
63650	%EX	Replaced by the locale's alternative time representation.
63651	%Ey	Replaced by the offset from %EC (year only) in the locale's alternative representation.
63652	%EY	Replaced by the full alternative year representation.
63653	%Od	Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero; otherwise, with leading <space> characters.
63654		
63655		
63656	%Oe	Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading <space> characters.
63657		
63658	%OH	Replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.
63659	%OI	Replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.
63660	%Om	Replaced by the month using the locale's alternative numeric symbols.
63661	%OM	Replaced by the minutes using the locale's alternative numeric symbols.

63662 %OS Replaced by the seconds using the locale's alternative numeric symbols.

63663 %Ou Replaced by the weekday as a number in the locale's alternative representation
63664 (Monday=1).

63665 %OU Replaced by the week number of the year (Sunday as the first day of the week, rules
63666 corresponding to %U) using the locale's alternative numeric symbols.

63667 %OV Replaced by the week number of the year (Monday as the first day of the week, rules
63668 corresponding to %V) using the locale's alternative numeric symbols.

63669 %Ow Replaced by the number of the weekday (Sunday=0) using the locale's alternative
63670 numeric symbols.

63671 %OW Replaced by the week number of the year (Monday as the first day of the week) using
63672 the locale's alternative numeric symbols.

63673 %Oy Replaced by the year (offset from %C) using the locale's alternative numeric symbols.

63674 %g, %G, and %V give values according to the ISO 8601:2004 standard week-based year. In this
63675 system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th,
63676 which is also the week that includes the first Thursday of the year, and is also the first week that
63677 contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the
63678 preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January
63679 1999, %G is replaced by 1998 and %V is replaced by 53. If December 29th, 30th, or 31st is a
63680 Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday
63681 30th December 1997, %G is replaced by 1998 and %v is replaced by 01.

63682 If a conversion specifier is not one of the above, the behavior is undefined.

RETURN VALUE

63683 If the total number of resulting bytes including the terminating null byte is not more than
63684 *maxsize*, these functions shall return the number of bytes placed into the array pointed to by *s*,
63685 not including the terminating NUL character. Otherwise, 0 shall be returned and the contents of
63686 the array are unspecified.
63687

ERRORS

63688 The *strftime_l()* function may fail if:

63690 CX [EINVAL] *locale* is not a valid locale object handle.

EXAMPLES**Getting a Localized Date String**

63692 The following example first sets the locale to the user's default. The locale information will be
63693 used in the *nl_langinfo()* and *strftime()* functions. The *nl_langinfo()* function returns the localized
63694 date string which specifies how the date is laid out. The *strftime()* function takes this
63695 information and, using the **tm** structure for values, places the date and time information into
63696 *datestring*.
63697

```
63698 #include <time.h>
63699 #include <locale.h>
63700 #include <langinfo.h>
63701 ...
63702 struct tm *tm;
63703 char datestring[256];
63704 ...
63705 setlocale (LC_ALL, "");
```


strftime()*System Interfaces*

```

63706     ...
63707     strftime (datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
63708     ...

```

APPLICATION USAGE

63710 The range of values for %S is [00,60] rather than [00,59] to allow for the occasional leap second.

63711 Some of the conversion specifications are duplicates of others. They are included for
 63712 compatibility with *nl_cxtime()* and *nl_ascxtime()*, which were published in Issue 2.

63713 The %C, %F, %G, and %Y format specifiers in *strftime()* always print full values, but the *strptime()*
 63714 %C, %F, and %Y format specifiers only scan two digits (assumed to be the first two digits of a
 63715 four-digit year) for %C and four digits (assumed to be the entire (four-digit) year) for %F and %Y.
 63716 This mimics the behavior of *printf()* and *scanf()*; that is:

```
63717 printf("%2d", x = 1000);
```

63718 prints "1000", but:

```
63719 scanf("%2d", &x);
```

63720 when given "1000" as input will only store 10 in x). Applications using extended ranges of
 63721 years must be sure that the number of digits specified for scanning years with *strptime()* matches
 63722 the number of digits that will actually be present in the input stream. Historic implementations
 63723 of the %Y conversion specification (with no flags and no minimum field width) produced
 63724 different output formats. Some always produced at least four digits (with 0 fill for years from 0
 63725 through 999) while others only produced the number of digits present in the year (with no fill
 63726 and no padding). These two forms can be produced with the '0' flag and a minimum field
 63727 width options using the conversions specifications %04Y and %01Y, respectively.

63728 In the past, the C and POSIX standards specified that %F produced an ISO 8601:2004 standard
 63729 date format, but didn't specify which one. For years in the range [0001,9999], POSIX.1-2008
 63730 requires that the output produced match the ISO 8601:2004 standard complete representation
 63731 extended format (YYYY-MM-DD) and for years outside of this range produce output that
 63732 matches the ISO 8601:2004 standard expanded representation extended format
 63733 (<+/-><Underline>Y</Underline>YYYY-MM-DD). To fully meet ISO 8601:2004 standard
 63734 requirements, the producer and consumer must agree on a date format that has a specific
 63735 number of bytes reserved to hold the characters used to represent the years that is sufficiently
 63736 large to hold all values that will be shared. For example, the %+13F conversion specification will
 63737 produce output matching the format "<+/->YYYYYY-MM-DD" (a leading '+' or '-' sign; a six-
 63738 digit, 0-filled year; a '-' ; a two-digit, leading 0-filled month; another '-' ; and the two-digit,
 63739 leading 0-filled day within the month).

63740 Note that if the year being printed is greater than 9999, the resulting string from the unadorned
 63741 %F conversion specifications will not conform to the ISO 8601:2004 standard extended format,
 63742 complete representation for a date and will instead be an extended format, expanded
 63743 representation (presumably without the required agreement between the date's producer and
 63744 consumer).

63745 In the C locale, the E and O modifiers are ignored and the replacement strings for the following
 63746 specifiers are:

63747 %a The first three characters of %A.

63748 %A One of Sunday, Monday, ..., Saturday.

63749	%b	The first three characters of %B.
63750	%B	One of January, February, ..., December.
63751	%c	Equivalent to %a %b %e %T %Y.
63752	%p	One of AM or PM.
63753	%r	Equivalent to %I:%M:%S %p.
63754	%x	Equivalent to %m/%d/%y.
63755	%X	Equivalent to %T.
63756	%Z	Implementation-defined.

RATIONALE

The %Y conversion specification to *strftime()* was frequently assumed to be a four-digit year, but the ISO C standard does not specify that %Y is restricted to any subset of allowed values from the *tm_year* field. Similarly, the %C conversion specification was assumed to be a two-digit field and the first part of the output from the %F conversion specification was assumed to be a four-digit field. With *tm_year* being a signed 32 or more-bit **int** and with many current implementations supporting 64-bit **time_t** types in one or more programming environments, these assumptions are clearly wrong.

POSIX.1-2008 now allows the format specifications %0xC, %0xF, %0xG, and %0xY (where 'x' is a string of decimal digits used to specify printing and scanning of a string of x decimal digits) with leading zero fill characters. Allowing applications to set the field width enables them to agree on the number of digits to be printed and scanned in the ISO 8601:2004 standard expanded representation of a year (for %F, %G, and %Y) or all but the last two digits of the year (for %C). This is based on a feature in some versions of GNU **libc**'s *strftime()*. The GNU version allows specifying space, zero, or no-fill characters in *strftime()* format strings, but does not allow any flags to be specified in *strftime()* format strings. These implementations also allow these flags to be specified for any numeric field. POSIX.1-2008 only requires the zero fill flag ('0') and only requires that it be recognized when processing %C, %F, %G, and %Y specifications when a minimum field width is also specified. The '0' flag is the only flag needed to produce and scan the ISO 8601:2004 standard year fields using the extended format forms. POSIX.1-2008 also allows applications to specify the same flag and field width specifiers to be used in both *strftime()* and *strptime()* format strings for symmetry. Systems may provide other flag characters and may accept flags in conjunction with conversion specifiers other than %C, %F, %G, and %Y; but portable applications cannot depend on such extensions.

POSIX.1-2008 now also allows the format specifications %+xC, %+xF, %+xG, and %+xY (where 'x' is a string of decimal digits used to specify printing and scanning of a string of 'x' decimal digits) with leading zero fill characters and a leading '+' sign character if the year being converted is more than four digits or a minimum field width is specified that allows room for more than four digits for the year. This allows date providers and consumers to agree on a specific number of digits to represent a year as required by the ISO 8601:2004 standard expanded representation formats. The expanded representation formats all require the year to begin with a leading '+' or '-' sign. (All of these specifiers can also provide a leading '-' sign for negative years. Since negative years and the year 0 don't fit well with the Gregorian or Julian calendars, the normal ranges of dates start with year 1. The ISO C standard allows *tm_year* to assume values corresponding to years before year 1, but the use of such years provided unspecified results.)

Some earlier version of this standard specified that applications wanting to use *strptime()* to scan dates and times printed by *strftime()* should provide non-digit characters between fields to

strftime()*System Interfaces*

separate years from months and days. It also supported %F to print and scan the ISO 8601:2004 standard extended format, complete representation date for years 1 through 9999 (i.e., YYYY-MM-DD). However, many applications were written to print (using *strftime()*) and scan (using *strptime()*) dates written using the basic format complete representation (four-digit years) and truncated representation (two-digit years) specified by the ISO 8601:2004 standard representation of dates and times which do not have any separation characters between fields. The ISO 8601:2004 standard also specifies basic format expanded representation where the creator and consumer of these fields agree beforehand to represent years as leading zero-filled strings of an agreed length of more than four digits to represent a year (again with no separation characters when year, month, and day are all displayed). Applications producing and consuming expanded representations are encouraged to use the '+' flag and an appropriate maximum field width to scan the year including the leading sign. Note that even without the '+' flag, years less than zero may be represented with a leading minus-sign for %F, %G, and %Y conversion specifications. Using negative years results in unspecified behavior.

If a format specification %+xF with the field width x greater than 11 is specified and the width is large enough to display the full year, the output string produced will match the ISO 8601:2004 standard subclause 4.1.2.4 expanded representation, extended format date representation for a specific day. (For years in the range [1,99 999], %+12F is sufficient for an agreed five-digit year with a leading sign using the ISO 8601:2004 standard expanded representation, extended format for a specific day "<+/->YYYY-MM-DD".) Note also that years less than 0 may produce a leading minus-sign ('-') when using %Y or %C whether or not the '0' or '+' flags are used.

The difference between the '0' flag and the '+' flag is whether the leading '+' character will be provided for years >9999 as required for the ISO 8601:2004 standard extended representation format containing a year. For example:

Year	Conversion Specification	<i>strftime()</i> Output	<i>strptime()</i> Scan Back
1970	%Y	1970	1970
1970	%+4Y	1970	1970
27	%Y	27 or 0027	27
270	%Y	270 or 0270	270
270	%+4Y	0270	270
17	%C%y	0017	17
270	%C%y	0270	270
12345	%Y	12345	1234*
12345	%+4Y	+12345	123*
12345	%05%Y	12345	12345
270	%+5Y or % +3C%y	+0270	270
12345	%+5Y or % +3C%y1+12345	1234*	
12345	%06Y or %04C%y	012345	12345
12345	%+6Y or % +4C%y	+12345	12345
123456	%08Y or %06C%y	00123456	123456
123456	%+8Y or % +6C%y	+0123456	123456

In the cases above marked with a * in the *strptime()* scan back field, the implied or specified number of characters scanned by *strptime()* was less than the number of characters output by *strftime()* using the same format; so the remaining digits of the year were dropped when the output date produced by *strftime()* was scanned back in by *strptime()*.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime(), *clock()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *time()*, *tzset()*, *uselocale()*, *utime()*

XBD Section 7.3.5 (on page 158), *<time.h>*

CHANGE HISTORY

First released in Issue 3.

Issue 5

The description of %OV is changed to be consistent with %V and defines Monday as the first day of the week.

The description of %Oy is clarified.

Issue 6

Extensions beyond the ISO C standard are marked.

The Open Group Corrigendum U033/8 is applied. The %V conversion specifier is changed from “Otherwise, it is week 53 of the previous year, and the next week is week 1” to “Otherwise, it is the last week of the previous year, and the next week is week 1”.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The %C, %D, %e, %h, %n, %r, %R, %t, and %T conversion specifiers are added.
- The modified conversion specifiers are added for consistency with the ISO POSIX-2 standard *date* utility.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *strftime()* prototype is updated.
- The DESCRIPTION is extensively revised.
- The %z conversion specifier is added.

An example is added.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/60 is applied.

Issue 7

Austin Group Interpretation 1003.1-2001 #163 is applied.

The *strftime_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

strlen()*System Interfaces*63873 **NAME**63874 **strlen, strlen** — get length of fixed size string63875 **SYNOPSIS**

63876 #include <string.h>

63877 size_t strlen(const char *s);

63878 CX size_t strlen(const char *s, size_t maxlen);

63879 **DESCRIPTION**

63880 CX For *strlen()*: The functionality described on this reference page is aligned with the ISO C
 63881 standard. Any conflict between the requirements described here and the ISO C standard is
 63882 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

63883 The *strlen()* function shall compute the number of bytes in the string to which *s* points, not
 63884 including the terminating NUL character.

63885 CX The *strlen()* function shall compute the smaller of the number of bytes in the array to which *s*
 63886 points, not including the terminating NUL character, or the value of the *maxlen* argument. The
 63887 *strlen()* function shall never examine more than *maxlen* bytes of the array pointed to by *s*.

63888 **RETURN VALUE**

63889 The *strlen()* function shall return the length of *s*; no return value shall be reserved to indicate an
 63890 error.

63891 CX The *strlen()* function shall return an integer containing the smaller of either the length of the
 63892 string pointed to by *s* or *maxlen*.

63893 **ERRORS**

63894 No errors are defined.

63895 **EXAMPLES**63896 **Getting String Lengths**

63897 The following example sets the maximum length of *key* and *data* by using *strlen()* to get the
 63898 lengths of those strings.

```

63899        #include <string.h>
63900        ...
63901        struct element {
63902            char *key;
63903            char *data;
63904        };
63905        ...
63906        char *key, *data;
63907        int len;

63908        *keylength = *datalength = 0;
63909        ...
63910        if ((len = strlen(key)) > *keylength)
63911            *keylength = len;
63912        if ((len = strlen(data)) > *datalength)
63913            *datalength = len;
63914        ...
```

63915 **APPLICATION USAGE**

63916 None.

63917 **RATIONALE**

63918 None.

63919 **FUTURE DIRECTIONS**

63920 None.

63921 **SEE ALSO**63922 *wcslen()*

63923 XBD <string.h>

63924 **CHANGE HISTORY**

63925 First released in Issue 1. Derived from Issue 1 of the SVID.

63926 **Issue 5**63927 The RETURN VALUE section is updated to indicate that *strlen()* returns the length of *s*, and not
63928 *s* itself as was previously stated.63929 **Issue 7**63930 The *strnlen()* function is added from The Open Group Technical Standard, 2006, Extended API
63931 Set Part 1.

strncasecmp()*System Interfaces*63932 **NAME**

63933 strncasecmp, strncasecmp_l — case-insensitive string comparisons

63934 **SYNOPSIS**

```
63935     #include <strings.h>
63936     int strncasecmp(const char *s1, const char *s2, size_t n);
63937     int strncasecmp_l(const char *s1, const char *s2,
63938                      size_t n, locale_t locale);
```

63939 **DESCRIPTION**63940 Refer to *strcasecmp()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

strncat — concatenate a string with part of another

SYNOPSIS

```
#include <string.h>
```

```
char *strncat(char *restrict s1, const char *restrict s2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *strncat()* function shall append not more than *n* bytes (a NUL character and bytes that follow it are not appended) from the array pointed to by *s2* to the end of the string pointed to by *s1*. The initial byte of *s2* overwrites the NUL character at the end of *s1*. A terminating NUL character is always appended to the result. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *strncat()* function shall return *s1*; no return value shall be reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[strcat\(\)](#)

XBD [<string.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The *strncat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

strncmp()

System Interfaces

63974 **NAME**63975 **strncmp** — compare part of two strings63976 **SYNOPSIS**63977 `#include <string.h>`63978 `int strncmp(const char *s1, const char *s2, size_t n);`63979 **DESCRIPTION**

63980 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 63981 conflict between the requirements described here and the ISO C standard is unintentional. This
 63982 volume of POSIX.1-2008 defers to the ISO C standard.

63983 The *strncmp()* function shall compare not more than *n* bytes (bytes that follow a NUL character
 63984 are not compared) from the array pointed to by *s1* to the array pointed to by *s2*.

63985 The sign of a non-zero return value is determined by the sign of the difference between the
 63986 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings
 63987 being compared.

63988 **RETURN VALUE**

63989 Upon successful completion, *strncmp()* shall return an integer greater than, equal to, or less than
 63990 0, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the
 63991 possibly null-terminated array pointed to by *s2* respectively.

63992 **ERRORS**

63993 No errors are defined.

63994 **EXAMPLES**

63995 None.

63996 **APPLICATION USAGE**

63997 None.

63998 **RATIONALE**

63999 None.

64000 **FUTURE DIRECTIONS**

64001 None.

64002 **SEE ALSO**64003 *strcmp()*64004 XBD *<string.h>*64005 **CHANGE HISTORY**

64006 First released in Issue 1. Derived from Issue 1 of the SVID.

64007 **Issue 6**

64008 Extensions beyond the ISO C standard are marked.

System Interfaces

strncpy()64009 **NAME**

64010 *stpncpy*, *strncpy* — copy fixed length string, returning a pointer to the array end

64011 **SYNOPSIS**

64012 `#include <string.h>`

64013 CX `char *stpncpy(char *restrict s1, const char *restrict s2, size_t n);`
 64014 `char *strncpy(char *restrict s1, const char *restrict s2, size_t n);`

64015 **DESCRIPTION**

64016 CX For *strncpy()*: The functionality described on this reference page is aligned with the ISO C
 64017 standard. Any conflict between the requirements described here and the ISO C standard is
 64018 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

64019 CX The *stpncpy()* and *strncpy()* functions shall copy not more than *n* bytes (bytes that follow a NUL
 64020 character are not copied) from the array pointed to by *s2* to the array pointed to by *s1*.

64021 If the array pointed to by *s2* is a string that is shorter than *n* bytes, NUL characters shall be
 64022 appended to the copy in the array pointed to by *s1*, until *n* bytes in all are written.

64023 If copying takes place between objects that overlap, the behavior is undefined.

64024 **RETURN VALUE**

64025 CX If a NUL character is written to the destination, the *stpncpy()* function shall return the address of
 64026 the first such NUL character. Otherwise, it shall return `&s1[n]`.

64027 The *strncpy()* function shall return *s1*.

64028 No return values are reserved to indicate an error.

64029 **ERRORS**

64030 No errors are defined.

64031 **EXAMPLES**

64032 None.

64033 **APPLICATION USAGE**

64034 Applications must provide the space in *s1* for the *n* bytes to be transferred, as well as ensure that
 64035 the *s2* and *s1* arrays do not overlap.

64036 Character movement is performed differently in different implementations. Thus, overlapping
 64037 moves may yield surprises.

64038 If there is no NUL character byte in the first *n* bytes of the array pointed to by *s2*, the result is not
 64039 null-terminated.

64040 **RATIONALE**

64041 None.

64042 **FUTURE DIRECTIONS**

64043 None.

64044 **SEE ALSO**

64045 *strcpy()*, *wcsncpy()*

64046 XBD `<string.h>`

64047 **CHANGE HISTORY**

64048 First released in Issue 1. Derived from Issue 1 of the SVID.

strncpy()*System Interfaces*64049 **Issue 6**64050 The *strncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.64051 **Issue 7**64052 The *strncpy()* function is added from The Open Group Technical Standard, 2006, Extended API
64053 Set Part 1.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***strndup()**64054 **NAME**

64055 strndup — duplicate a specific number of bytes from a string

64056 **SYNOPSIS**

64057 CX #include <string.h>

64058 char *strndup(const char *s, size_t size);

64059 **DESCRIPTION**64060 Refer to *strdup()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

strnlen()*System Interfaces*64061 **NAME**

64062 strnlen — get length of fixed size string

64063 **SYNOPSIS**

64064 CX #include <string.h>

64065 size_t strnlen(const char *s, size_t maxlen);

64066 **DESCRIPTION**64067 Refer to *strlen()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

strpbrk()**NAME**

strpbrk — scan a string for a byte

SYNOPSIS

```
#include <string.h>
```

```
char *strpbrk(const char *s1, const char *s2);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *strpbrk()* function shall locate the first occurrence in the string pointed to by *s1* of any byte from the string pointed to by *s2*.

RETURN VALUE

Upon successful completion, *strpbrk()* shall return a pointer to the byte or a null pointer if no byte from *s2* occurs in *s1*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strchr(), *strrchr()*

XBD *<string.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

strptime()*System Interfaces*64097 **NAME**64098 **strptime** — date and time conversion64099 **SYNOPSIS**

```
64100 XSI #include <time.h>
64101 char *strptime(const char *restrict buf, const char *restrict format,
64102               struct tm *restrict tm);
```

64103 **DESCRIPTION**

64104 The *strptime()* function shall convert the character string pointed to by *buf* to values which are
 64105 stored in the **tm** structure pointed to by *tm*, using the format specified by *format*.

64106 The format is composed of zero or more directives. Each directive is composed of one of the
 64107 following: one or more white-space characters (as specified by *isspace()*); an ordinary character
 64108 (neither `' % '` nor a white-space character); or a conversion specification.

64109 Each conversion specification is introduced by the `' % '` character after which the following
 64110 appear in sequence:

- 64111 • An optional flag, the zero character (`' 0 '`) or the `<plus-sign>` character (`' + '`), which is
 64112 ignored.
- 64113 • An optional field width. If a field width is specified, it shall be interpreted as a string of
 64114 decimal digits that will determine the maximum number of bytes converted for the
 64115 conversion rather than the number of bytes specified below in the description of the
 64116 conversion specifiers.
- 64117 • An optional `E` or `O` modifier.
- 64118 • A terminating conversion specifier character that indicates the type of conversion to be
 64119 applied.

64120 The conversions are determined using the *LC_TIME* category of the current locale. The
 64121 application shall ensure that there is white-space or other non-alphanumeric characters between
 64122 any two conversion specifications unless all of the adjacent conversion specifications convert a
 64123 known, fixed number of characters. In the following list, the maximum number of characters
 64124 scanned (excluding the one matching the next directive) is as follows:

- 64125 • If a maximum field width is specified, then that number
- 64126 • Otherwise, the pattern `" { x } "` indicates that the maximum is *x*
- 64127 • Otherwise, the pattern `" [ x, y ] "` indicates that the value shall fall within the range given
 64128 (both bounds being inclusive), and the maximum number of characters scanned shall be
 64129 the maximum required to represent any value in the range without leading zeros and
 64130 without a leading `<plus-sign>`

64131 The following conversion specifiers are supported.

64132 The results are unspecified if a modifier is specified with a flag or with a minimum field width,
 64133 or if a field width is specified for any conversion specifier other than `C`, `E`, or `Y`.

64134	a	The day of the week, using the locale's weekday names; either the abbreviated or full
64135		name may be specified.
64136	A	Equivalent to <code>%a</code> .

System Interfaces

strptime()

64137	b	The month, using the locale's month names; either the abbreviated or full name may be specified.
64138		
64139	B	Equivalent to %b.
64140	c	Replaced by the locale's appropriate date and time representation.
64141	C	All but the last two digits of the year {2}; leading zeros shall be permitted but shall not be required. A leading '+' or '-' character shall be permitted before any leading zeros but shall not be required.
64142		
64143		
64144	d	The day of the month [01,31]; leading zeros shall be permitted but shall not be required.
64145	D	The date as %m/%d/%y.
64146	e	Equivalent to %d.
64147	h	Equivalent to %b.
64148	H	The hour (24-hour clock) [00,23]; leading zeros shall be permitted but shall not be required.
64149		
64150	I	The hour (12-hour clock) [01,12]; leading zeros shall be permitted but shall not be required.
64151		
64152	j	The day number of the year [001,366]; leading zeros shall be permitted but shall not be required.
64153		
64154	m	The month number [01,12]; leading zeros shall be permitted but shall not be required.
64155	M	The minute [00,59]; leading zeros shall be permitted but shall not be required.
64156	n	Any white space.
64157	p	The locale's equivalent of a.m. or p.m.
64158	r	12-hour clock time using the AM/PM notation if <code>t_fmt_ampm</code> is not an empty string in the <code>LC_TIME</code> portion of the current locale; in the POSIX locale, this shall be equivalent to %I:%M:%S %p.
64159		
64160		
64161	R	The time as %H:%M.
64162	S	The seconds [00,60]; leading zeros shall be permitted but shall not be required.
64163	t	Any white space.
64164	T	The time as %H:%M:%S.
64165	U	The week number of the year (Sunday as the first day of the week) as a decimal number [00,53]; leading zeros shall be permitted but shall not be required.
64166		
64167	w	The weekday as a decimal number [0,6], with 0 representing Sunday.
64168	W	The week number of the year (Monday as the first day of the week) as a decimal number [00,53]; leading zeros shall be permitted but shall not be required.
64169		
64170	x	The date, using the locale's date format.
64171	X	The time, using the locale's time format.
64172	y	The last two digits of the year. When <i>format</i> contains neither a C conversion specifier nor a Y conversion specifier, values in the range [69,99] shall refer to years 1969 to 1999 inclusive and values in the range [00,68] shall refer to years 2000 to 2068 inclusive; leading zeros shall be permitted but shall not be required. A leading '+' or '-'
64173		
64174		
64175		

strptime()*System Interfaces*

- 64176 character shall be permitted before any leading zeros but shall not be required.
- 64177 **Note:** It is expected that in a future version of this standard the default century inferred
 64178 from a 2-digit year will change. (This would apply to all commands accepting a
 64179 2-digit year as input.)
- 64180 **Y** The full year {4}; leading zeros shall be permitted but shall not be required. A leading
 64181 '+' or '-' character shall be permitted before any leading zeros but shall not be
 64182 required.
- 64183 **%** Replaced by %.

Modified Conversion Specifiers

64185 Some conversion specifiers can be modified by the **E** and **O** modifier characters to indicate that
 64186 an alternative format or specification should be used rather than the one normally used by the
 64187 unmodified conversion specifier. If the alternative format or specification does not exist in the
 64188 current locale, the behavior shall be as if the unmodified conversion specification were used.

- 64189 **%Ec** The locale's alternative appropriate date and time representation.
- 64190 **%EC** The name of the base year (period) in the locale's alternative representation.
- 64191 **%Ex** The locale's alternative date representation.
- 64192 **%EX** The locale's alternative time representation.
- 64193 **%Ey** The offset from **%EC** (year only) in the locale's alternative representation.
- 64194 **%EY** The full alternative year representation.
- 64195 **%Od** The day of the month using the locale's alternative numeric symbols; leading zeros
 64196 shall be permitted but shall not be required.
- 64197 **%Oe** Equivalent to **%Od**.
- 64198 **%OH** The hour (24-hour clock) using the locale's alternative numeric symbols.
- 64199 **%OI** The hour (12-hour clock) using the locale's alternative numeric symbols.
- 64200 **%Om** The month using the locale's alternative numeric symbols.
- 64201 **%OM** The minutes using the locale's alternative numeric symbols.
- 64202 **%OS** The seconds using the locale's alternative numeric symbols.
- 64203 **%OU** The week number of the year (Sunday as the first day of the week) using the locale's
 64204 alternative numeric symbols.
- 64205 **%Ow** The number of the weekday (Sunday=0) using the locale's alternative numeric
 64206 symbols.
- 64207 **%OW** The week number of the year (Monday as the first day of the week) using the locale's
 64208 alternative numeric symbols.
- 64209 **%Oy** The year (offset from **%C**) using the locale's alternative numeric symbols.

64210 A conversion specification composed of white-space characters is executed by scanning input up
 64211 to the first character that is not white-space (which remains unscanned), or until no more
 64212 characters can be scanned.

64213 A conversion specification that is an ordinary character is executed by scanning the next
 64214 character from the buffer. If the character scanned from the buffer differs from the one
 64215 comprising the directive, the directive fails, and the differing and subsequent characters remain

unscanned.

A series of conversion specifications composed of %n, %t, white-space characters, or any combination is executed by scanning up to the first character that is not white space (which remains unscanned), or until no more characters can be scanned.

Any other conversion specification is executed by scanning characters until a character matching the next directive is scanned, or until no more characters can be scanned. These characters, except the one matching the next directive, are then compared to the locale values associated with the conversion specifier. If a match is found, values for the appropriate **tm** structure members are set to values corresponding to the locale information. Case is ignored when matching items in *buf* such as month or weekday names. If no match is found, *strptime()* fails and no more characters are scanned.

RETURN VALUE

Upon successful completion, *strptime()* shall return a pointer to the character following the last character parsed. Otherwise, a null pointer shall be returned.

ERRORS

No errors are defined.

EXAMPLES

Convert a Data-Plus-Time String to Broken-Down Time and Then into Seconds

The following example demonstrates the use of *strptime()* to convert a string into broken-down time. The broken-down time is then converted into seconds since the Epoch using *mktime()*.

```
#include <time.h>
...

struct tm tm;
time_t t;

if (strptime("6 Dec 2001 12:33:45", "%d %b %Y %H:%M:%S", &tm) == NULL)
    /* Handle error */;

printf("year: %d; month: %d; day: %d;\n",
       tm.tm_year, tm.tm_mon, tm.tm_mday);
printf("hour: %d; minute: %d; second: %d\n",
       tm.tm_hour, tm.tm_min, tm.tm_sec);
printf("week day: %d; year day: %d\n", tm.tm_wday, tm.tm_yday);

tm.tm_isdst = -1;          /* Not set by strptime(); tells mktime()
                           to determine whether daylight saving time
                           is in effect */

t = mktime(&tm);
if (t == -1)
    /* Handle error */;

printf("seconds since the Epoch: %ld\n", (long) t);"
```

APPLICATION USAGE

Several "equivalent to" formats and the special processing of white-space characters are provided in order to ease the use of identical *format* strings for *strptime()* and *strptime()*.

It should be noted that dates constructed by the *strptime()* function with the %Y or %C%y conversion specifiers may have values larger than 9999. If the *strptime()* function is used to read such values using %C%y or %Y, the year values will be truncated to four digits. Applications

strptime()*System Interfaces*

- 64260 should use `%+w%Y` or `%+xY` with *w* and *x* set large enough to contain the full value of any years
64261 that will be printed or scanned.
- 64262 See also the APPLICATION USAGE section in *strptime()*.
- 64263 It is unspecified whether multiple calls to *strptime()* using the same **tm** structure will update the
64264 current contents of the structure or overwrite all contents of the structure. Conforming
64265 applications should make a single call to *strptime()* with a format and all data needed to
64266 completely specify the date and time being converted.
- 64267 **RATIONALE**
- 64268 See the RATIONALE section for *strptime()*.
- 64269 **FUTURE DIRECTIONS**
- 64270 None.
- 64271 **SEE ALSO**
- 64272 *fprintf()*, *fscanf()*, *strptime()*, *time()*
- 64273 XBD **<time.h>**
- 64274 **CHANGE HISTORY**
- 64275 First released in Issue 4.
- 64276 **Issue 5**
- 64277 Moved from ENHANCED I18N to BASE.
- 64278 The [ENOSYS] error is removed.
- 64279 The exact meaning of the `%y` and `%0y` specifiers is clarified in the DESCRIPTION.
- 64280 **Issue 6**
- 64281 The Open Group Corrigendum U033/5 is applied. The `%r` specifier description is reworded.
- 64282 The normative text is updated to avoid use of the term “must” for application requirements.
- 64283 The **restrict** keyword is added to the *strptime()* prototype for alignment with the
64284 ISO/IEC 9899:1999 standard.
- 64285 The Open Group Corrigendum U047/2 is applied.
- 64286 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion
64287 specification” for consistency with *strptime()*.
- 64288 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/133 is applied, adding the example to the
64289 EXAMPLES section.
- 64290 **Issue 7**
- 64291 Austin Group Interpretation 1003.1-2001 #041 is applied, updating the DESCRIPTION and
64292 APPLICATION USAGE sections.
- 64293 Austin Group Interpretation 1003.1-2001 #163 is applied.
- 64294 SD5-XSH-ERN-67 is applied, correcting the APPLICATION USAGE to remove the impression
64295 that `%Y` is 4-digit years.

NAME

strrchr — string scanning operation

SYNOPSIS

#include <string.h>

char *strrchr(const char *s, int c);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *strrchr()* function shall locate the last occurrence of *c* (converted to a **char**) in the string pointed to by *s*. The terminating NUL character is considered to be part of the string.

RETURN VALUE

Upon successful completion, *strrchr()* shall return a pointer to the byte or a null pointer if *c* does not occur in the string.

ERRORS

No errors are defined.

EXAMPLES**Finding the Base Name of a File**

The following example uses *strrchr()* to get a pointer to the base name of a file. The *strrchr()* function searches backwards through the name of the file to find the last '/' character in *name*. This pointer (plus one) will point to the base name of the file.

```
#include <string.h>
...
const char *name;
char *basename;
...
basename = strrchr(name, '/') + 1;
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strchr()

XBD <string.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

strsignal()

System Interfaces

64335 **NAME**64336 **strsignal** — get name of signal64337 **SYNOPSIS**

```
64338 CX    #include <string.h>
64339        char *strsignal(int signum);
```

64340 **DESCRIPTION**

64341 The *strsignal()* function shall map the signal number in *signum* to an implementation-defined string and shall return a pointer to it. It shall use the same set of messages as the *psignal()* function.

64344 The string pointed to shall not be modified by the application, but may be overwritten by a subsequent call to *strsignal()* or *setlocale()*.

64346 The contents of the message strings returned by *strsignal()* should be determined by the setting of the *LC_MESSAGES* category in the current locale.

64348 The implementation shall behave as if no function defined in this standard calls *strsignal()*.

64349 Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strsignal()*, then check *errno*.

64351 The *strsignal()* function need not be thread-safe.

64352 **RETURN VALUE**

64353 Upon successful completion, *strsignal()* shall return a pointer to a string. Otherwise, if *signum* is not a valid signal number, the return value is unspecified.

64355 **ERRORS**

64356 No errors are defined.

64357 **EXAMPLES**

64358 None.

64359 **APPLICATION USAGE**

64360 None.

64361 **RATIONALE**

64362 If *signum* is not a valid signal number, some implementations return NULL, while for others the *strsignal()* function returns a pointer to a string containing an unspecified message denoting an unknown signal. POSIX.1-2008 leaves this return value unspecified.

64365 **FUTURE DIRECTIONS**

64366 None.

64367 **SEE ALSO**

64368 *psiginfo()*, *setlocale()*

64369 XBD *<string.h>*

64370 **CHANGE HISTORY**

64371 First released in Issue 7.

64372 **NAME**

64373 strspn — get length of a substring

64374 **SYNOPSIS**

64375 #include <string.h>

64376 size_t strspn(const char *s1, const char *s2);

64377 **DESCRIPTION**

64378 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 64379 conflict between the requirements described here and the ISO C standard is unintentional. This
 64380 volume of POSIX.1-2008 defers to the ISO C standard.

64381 The *strspn()* function shall compute the length (in bytes) of the maximum initial segment of the
 64382 string pointed to by *s1* which consists entirely of bytes from the string pointed to by *s2*.

64383 **RETURN VALUE**

64384 The *strspn()* function shall return the computed length; no return value is reserved to indicate an
 64385 error.

64386 **ERRORS**

64387 No errors are defined.

64388 **EXAMPLES**

64389 None.

64390 **APPLICATION USAGE**

64391 None.

64392 **RATIONALE**

64393 None.

64394 **FUTURE DIRECTIONS**

64395 None.

64396 **SEE ALSO**64397 *strcspn()*

64398 XBD <string.h>

64399 **CHANGE HISTORY**

64400 First released in Issue 1. Derived from Issue 1 of the SVID.

64401 **Issue 5**

64402 The RETURN-VALUE section is updated to indicate that *strspn()* returns the length of *s*, and not
 64403 *s* itself as was previously stated.

64404 **Issue 7**

64405 SD5-XSH-ERN-182 is applied.

strstr()

System Interfaces

64406 **NAME**64407 **strstr** — find a substring64408 **SYNOPSIS**64409 `#include <string.h>`64410 `char *strstr(const char *s1, const char *s2);`64411 **DESCRIPTION**

64412 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
64413 conflict between the requirements described here and the ISO C standard is unintentional. This
64414 volume of POSIX.1-2008 defers to the ISO C standard.

64415 The *strstr()* function shall locate the first occurrence in the string pointed to by *s1* of the
64416 sequence of bytes (excluding the terminating NUL character) in the string pointed to by *s2*.

64417 **RETURN VALUE**

64418 Upon successful completion, *strstr()* shall return a pointer to the located string or a null pointer
64419 if the string is not found.

64420 If *s2* points to a string with zero length, the function shall return *s1*.

64421 **ERRORS**

64422 No errors are defined.

64423 **EXAMPLES**

64424 None.

64425 **APPLICATION USAGE**

64426 None.

64427 **RATIONALE**

64428 None.

64429 **FUTURE DIRECTIONS**

64430 None.

64431 **SEE ALSO**

64432 [*strchr\(\)*](#)

64433 XBD [*<string.h>*](#)

64434 **CHANGE HISTORY**

64435 First released in Issue 3. Included for alignment with the ANSI C standard.

NAME

strtod, strtodf, strtold — convert a string to a double-precision number

SYNOPSIS

```
#include <stdlib.h>
```

```
double strtod(const char *restrict nptr, char **restrict endptr);
```

```
float strtodf(const char *restrict nptr, char **restrict endptr);
```

```
long double strtold(const char *restrict nptr, char **restrict endptr);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall convert the initial portion of the string pointed to by *nptr* to **double**, **float**, and **long double** representation, respectively. First, they decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
3. A final string of one or more unrecognized characters, including the terminating NUL character of the input string

Then they shall attempt to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional '+' or '-' sign, then one of the following:

- A non-empty sequence of decimal digits optionally containing a radix character; then an optional exponent part consisting of the character 'e' or the character 'E', optionally followed by a '+' or '-' character, and then followed by one or more decimal digits
- A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix character; then an optional binary exponent part consisting of the character 'p' or the character 'P', optionally followed by a '+' or '-' character, and then followed by one or more decimal digits
- One of INF or INFINITY, ignoring case
- One of NAN or NAN(*n-char-sequence_{opt}*), ignoring case in the NAN part, where:

n-char-sequence:

digit

nondigit

n-char-sequence digit

n-char-sequence nondigit

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is not of the expected form.

If the subject sequence has the expected form for a floating-point number, the sequence of characters starting with the first digit or the decimal-point character (whichever occurs first) shall be interpreted as a floating constant of the C language, except that the radix character shall be used in place of a period, and that if neither an exponent part nor a radix character appears in

strtod()*System Interfaces*

64480		a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal
64481		floating-point number, an exponent part of the appropriate type with value zero is assumed to
64482		follow the last digit in the string. If the subject sequence begins with a minus-sign, the sequence
64483		shall be interpreted as negated. A character sequence INF or INFINITY shall be interpreted as an
64484		infinity, if representable in the return type, else as if it were a floating constant that is too large
64485		for the range of the return type. A character sequence NAN or NAN(<i>n-char-sequence_{opt}</i>) shall be
64486		interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence
64487		part that does not have the expected form; the meaning of the <i>n-char</i> sequences is
64488		implementation-defined. A pointer to the final string is stored in the object pointed to by <i>endptr</i> ,
64489		provided that <i>endptr</i> is not a null pointer.
64490		If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the value
64491		resulting from the conversion is correctly rounded.
64492	CX	The radix character is defined in the locale of the process (category <i>LC_NUMERIC</i>). In the
64493		POSIX locale, or in a locale where the radix character is not defined, the radix character shall
64494		default to a <period> (' . ').
64495	CX	In other than the C or POSIX locales, other implementation-defined subject sequences may be
64496		accepted.
64497		If the subject sequence is empty or does not have the expected form, no conversion shall be
64498		performed; the value of <i>str</i> is stored in the object pointed to by <i>endptr</i> , provided that <i>endptr</i> is not
64499		a null pointer.
64500	CX	The <i>strtod()</i> function shall not change the setting of <i>errno</i> if successful.
64501		Since 0 is returned on error and is also a valid return on success, an application wishing to check
64502		for error situations should set <i>errno</i> to 0, then call <i>strtod()</i> , <i>strtof()</i> , or <i>strtold()</i> , then check <i>errno</i> .
64503	RETURN VALUE	
64504		Upon successful completion, these functions shall return the converted value. If no conversion
64505		could be performed, 0 shall be returned, and <i>errno</i> may be set to [EINVAL].
64506		If the correct value is outside the range of representable values, ±HUGE_VAL, ±HUGE_VALF, or
64507		±HUGE_VALL shall be returned (according to the sign of the value), and <i>errno</i> shall be set to
64508		[ERANGE].
64509		If the correct value would cause an underflow, a value whose magnitude is no greater than the
64510		smallest normalized positive number in the return type shall be returned and <i>errno</i> set to
64511		[ERANGE].
64512	ERRORS	
64513		These functions shall fail if:
64514	CX	[ERANGE] The value to be returned would cause overflow or underflow.
64515		These functions may fail if:
64516	CX	[EINVAL] No conversion could be performed.

EXAMPLES

None.

APPLICATION USAGE

If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the result is not exactly representable, the result should be one of the two numbers in the appropriate internal format that are adjacent to the hexadecimal floating source value, with the extra stipulation that the error should have a correct sign for the current rounding direction.

If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in <float.h>) significant digits, the result should be correctly rounded. If the subject sequence *D* has the decimal form and more than DECIMAL_DIG significant digits, consider the two bounding, adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the values of *L*, *D*, and *U* satisfy $L \leq D \leq U$. The result should be one of the (equal or adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current rounding direction, with the extra stipulation that the error with respect to *D* should have a correct sign for the current rounding direction.

The changes to *strtod()* introduced by the ISO/IEC 9899:1999 standard can alter the behavior of well-formed applications complying with the ISO/IEC 9899:1990 standard and thus earlier versions of this standard. One such example would be:

```
int
what_kind_of_number (char *s)
{
    char *endp;
    double d;
    long l;

    d = strtod(s, &endp);
    if (s != endp && *endp == '\0')
        printf("It's a float with value %g\n", d);
    else
    {
        l = strtol(s, &endp, 0);
        if (s != endp && *endp == '\0')
            printf("It's an integer with value %ld\n", l);
        else
            return 1;
    }
    return 0;
}
```

If the function is called with:

```
what_kind_of_number ("0x10")
```

an ISO/IEC 9899:1990 standard-compliant library will result in the function printing:

```
It's an integer with value 16
```

With the ISO/IEC 9899:1999 standard, the result is:

```
It's a float with value 16
```

The change in behavior is due to the inclusion of floating-point numbers in hexadecimal notation without requiring that either a decimal point or the binary exponent be present.

strtod()*System Interfaces*64562 **RATIONALE**

64563 None.

64564 **FUTURE DIRECTIONS**

64565 None.

64566 **SEE ALSO**64567 *fscanf()*, *isspace()*, *localeconv()*, *setlocale()*, *strtol()*64568 XBD Chapter 7 (on page 135), *<float.h>*, *<stdlib.h>*64569 **CHANGE HISTORY**

64570 First released in Issue 1. Derived from Issue 1 of the SVID.

64571 **Issue 5**64572 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.64573 **Issue 6**

64574 Extensions beyond the ISO C standard are marked.

64575 The following new requirements on POSIX implementations derive from alignment with the
64576 Single UNIX Specification:

- 64577 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
- 64578 added if no conversion could be performed.

64579 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 64580 • The *strtod()* function is updated.
- 64581 • The *strtodf()* and *strtold()* functions are added.
- 64582 • The DESCRIPTION is extensively revised.

64583 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

64584 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/61 is applied, correcting the second
64585 paragraph in the RETURN VALUE section. This change clarifies the sign of the return value.64586 **Issue 7**

64587 Austin Group Interpretation 1003.1-2001 #015 is applied.

NAME

strtoimax, strtoumax — convert string to integer type

SYNOPSIS

```
#include <inttypes.h>

intmax_t strtoimax(const char *restrict nptr, char **restrict endptr,
    int base);
uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,
    int base);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall be equivalent to the *strtol()*, *strtoll()*, *strtoul()*, and *strtoull()* functions, except that the initial portion of the string shall be converted to **intmax_t** and **uintmax_t** representation, respectively.

RETURN VALUE

These functions shall return the converted value, if any.

If no conversion could be performed, zero shall be returned.

If the correct value is outside the range of representable values, {INTMAX_MAX}, {INTMAX_MIN}, or {UINTMAX_MAX} shall be returned (according to the return type and sign of the value, if any), and *errno* shall be set to [ERANGE].

ERRORS

These functions shall fail if:

[ERANGE] The value to be returned is not representable.

These functions may fail if:

[EINVAL] The value of *base* is not supported.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtol(), *strtoul()*

XBD **<inttypes.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

strtok()*System Interfaces*64627 **NAME**64628 **strtok, strtok_r** — split string into tokens64629 **SYNOPSIS**64630 `#include <string.h>`64631 `char *strtok(char *restrict s1, const char *restrict s2);`64632 CX `char *strtok_r(char *restrict s, const char *restrict sep,`
64633 `char **restrict lasts);`64634 **DESCRIPTION**64635 CX For *strtok()*: The functionality described on this reference page is aligned with the ISO C
64636 standard. Any conflict between the requirements described here and the ISO C standard is
64637 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.64638 A sequence of calls to *strtok()* breaks the string pointed to by *s1* into a sequence of tokens, each
64639 of which is delimited by a byte from the string pointed to by *s2*. The first call in the sequence
64640 has *s1* as its first argument, and is followed by calls with a null pointer as their first argument.
64641 The separator string pointed to by *s2* may be different from call to call.64642 The first call in the sequence searches the string pointed to by *s1* for the first byte that is *not*
64643 contained in the current separator string pointed to by *s2*. If no such byte is found, then there
64644 are no tokens in the string pointed to by *s1* and *strtok()* shall return a null pointer. If such a byte
64645 is found, it is the start of the first token.64646 The *strtok()* function then searches from there for a byte that *is* contained in the current separator
64647 string. If no such byte is found, the current token extends to the end of the string pointed to by
64648 *s1*, and subsequent searches for a token shall return a null pointer. If such a byte is found, it is
64649 overwritten by a NUL character, which terminates the current token. The *strtok()* function saves
64650 a pointer to the following byte, from which the next search for a token shall start.64651 Each subsequent call, with a null pointer as the value of the first argument, starts searching from
64652 the saved pointer and behaves as described above.64653 The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls
64654 *strtok()*.64655 CX The *strtok()* function need not be thread-safe.64656 The *strtok_r()* function considers the null-terminated string *s* as a sequence of zero or more text
64657 tokens separated by spans of one or more characters from the separator string *sep*. The
64658 argument *lasts* points to a user-provided pointer which points to stored information necessary
64659 for *strtok_r()* to continue scanning the same string.64660 In the first call to *strtok_r()*, *s* points to a null-terminated string, *sep* to a null-terminated string of
64661 separator characters, and the value pointed to by *lasts* is ignored. The *strtok_r()* function shall
64662 return a pointer to the first character of the first token, write a null character into *s* immediately
64663 following the returned token, and update the pointer to which *lasts* points.64664 In subsequent calls, *s* is a null pointer and *lasts* shall be unchanged from the previous call so that
64665 subsequent calls shall move through the string *s*, returning successive tokens until no tokens
64666 remain. The separator string *sep* may be different from call to call. When no token remains in *s*, a
64667 null pointer shall be returned.

RETURN VALUE

Upon successful completion, *strtok()* shall return a pointer to the first byte of a token. Otherwise, if there is no token, *strtok()* shall return a null pointer.

CX

The *strtok_r()* function shall return a pointer to the token found, or a null pointer when no token is found.

ERRORS

No errors are defined.

EXAMPLES**Searching for Word Separators**

The following example searches for tokens separated by <space> characters.

```
#include <string.h>
...
char *token;
char line[] = "LINE TO BE SEPARATED";
char *search = " ";

/* Token will point to "LINE". */
token = strtok(line, search);

/* Token will point to "TO". */
token = strtok(NULL, search);
```

Breaking a Line

The following example uses *strtok()* to break a line into two character strings separated by any combination of <space>, <tab>, or <newline> characters.

```
#include <string.h>
...
struct element {
    char *key;
    char *data;
};
...
char line[LINE_MAX];
char *key, *data;
...
key = strtok(line, " \n");
data = strtok(NULL, " \n");
...
```

APPLICATION USAGE

The *strtok_r()* function is thread-safe and stores its state in a user-supplied buffer instead of possibly using a static data area that may be overwritten by an unrelated call from another thread.

RATIONALE

The *strtok()* function searches for a separator string within a larger string. It returns a pointer to the last substring between separator strings. This function uses static storage to keep track of the current string position between calls. The new function, *strtok_r()*, takes an additional argument, *lasts*, to keep track of the current position in the string.

strtok()*System Interfaces*64712 **FUTURE DIRECTIONS**

64713 None.

64714 **SEE ALSO**64715 XBD [<string.h>](#)64716 **CHANGE HISTORY**

64717 First released in Issue 1. Derived from Issue 1 of the SVID.

64718 **Issue 5**64719 The *strtok_r()* function is included for alignment with the POSIX Threads Extension.64720 A note indicating that the *strtok()* function need not be reentrant is added to the DESCRIPTION.64721 **Issue 6**

64722 Extensions beyond the ISO C standard are marked.

64723 The *strtok_r()* function is marked as part of the Thread-Safe Functions option.

64724 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

64725 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
64726 its avoidance of possibly using a static data area.64727 The **restrict** keyword is added to the *strtok()* and *strtok_r()* prototypes for alignment with the
64728 ISO/IEC 9899:1999 standard.64729 **Issue 7**

64730 Austin Group Interpretation 1003.1-2001 #156 is applied.

64731 SD5-XSH-ERN-235 is applied, correcting an example.

64732 The *strtok_r()* function is moved from the Thread-Safe Functions option to the Base.

NAME

strtol, strtoll — convert a string to a long integer

SYNOPSIS

```
#include <stdlib.h>
```

```
long strtol(const char *restrict str, char **restrict endptr, int base);
long long strtoll(const char *restrict str, char **restrict endptr,
int base)
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall convert the initial portion of the string pointed to by *str* to a type **long** and **long long** representation, respectively. First, they decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final string of one or more unrecognized characters, including the terminating NUL character of the input string.

Then they shall attempt to convert the subject sequence to an integer, and return the result.

If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character that is of the expected form. The subject sequence shall contain no characters if the input string is empty or consists entirely of white-space characters, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is 0, the sequence of characters starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus-sign, the value resulting from the conversion shall be negated. A pointer to the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the C or POSIX locales, other implementation-defined subject sequences may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed;

strtol()*System Interfaces*

64778 the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null
 64779 pointer.

64780 CX The *strtol()* function shall not change the setting of *errno* if successful.

64781 Since 0, {LONG_MIN} or {LLONG_MIN}, and {LONG_MAX} or {LLONG_MAX} are returned
 64782 on error and are also valid returns on success, an application wishing to check for error
 64783 situations should set *errno* to 0, then call *strtol()* or *strtoll()*, then check *errno*.

RETURN VALUE

64785 Upon successful completion, these functions shall return the converted value, if any. If no
 64786 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

64787 If the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX},
 64788 {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and
 64789 *errno* set to [ERANGE].

ERRORS

64790 These functions shall fail if:

64791 [ERANGE] The value to be returned is not representable.

64792 These functions may fail if:

64793 [EINVAL] The value of *base* is not supported.

EXAMPLES

64795 None.

APPLICATION USAGE

64796 None.

RATIONALE

64797 None.

FUTURE DIRECTIONS

64800 None.

SEE ALSO

64803 *fscanf()*, *isalpha()*, *strtod()*

64804 XBD <stdlib.h>

CHANGE HISTORY

64806 First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

64807 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

64808 Extensions beyond the ISO C standard are marked.

64812 The following new requirements on POSIX implementations derive from alignment with the
 64813 Single UNIX Specification:

- 64814 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
 64815 added if no conversion could be performed.

64816

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

64817

- The *strtol()* prototype is updated.

64818

- The *strtoll()* function is added.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

strtold()*System Interfaces*64819 **NAME**64820 **strtold** — convert a string to a double-precision number64821 **SYNOPSIS**64822 `#include <stdlib.h>`64823 `long double strtold(const char *restrict nptr, char **restrict endptr);`64824 **DESCRIPTION**64825 Refer to *strtod()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***strtoll()**64826 **NAME**64827 **strtoll** — convert a string to a long integer64828 **SYNOPSIS**64829 `#include <stdlib.h>`64830 `long long strtoll(const char *restrict str, char **restrict endptr,`
64831 `int base);`64832 **DESCRIPTION**64833 Refer to *strtol()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

strtoul()*System Interfaces*64834 **NAME**64835 **strtoul, strtoull** — convert a string to an unsigned long64836 **SYNOPSIS**64837 `#include <stdlib.h>`64838 `unsigned long strtoul(const char *restrict str,`
64839 `char **restrict endptr, int base);`64840 `unsigned long long strtoull(const char *restrict str,`
64841 `char **restrict endptr, int base);`64842 **DESCRIPTION**64843 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
64844 conflict between the requirements described here and the ISO C standard is unintentional. This
64845 volume of POSIX.1-2008 defers to the ISO C standard.64846 These functions shall convert the initial portion of the string pointed to by *str* to a type **unsigned**
64847 **long** and **unsigned long long** representation, respectively. First, they decompose the input string
64848 into three parts:

- 64849 1. An initial, possibly empty, sequence of white-space characters (as specified by
- isspace()*
-)
-
- 64850 2. A subject sequence interpreted as an integer represented in some radix determined by the
-
- 64851 value of
- base*
-
- 64852 3. A final string of one or more unrecognized characters, including the terminating NUL
-
- 64853 character of the input string

64854 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the
64855 result.64856 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,
64857 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A
64858 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An
64859 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to
64860 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
64861 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.64862 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
64863 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
64864 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the
64865 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the
64866 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and
64867 digits, following the sign if present.64868 The subject sequence is defined as the longest initial subsequence of the input string, starting
64869 with the first non-white-space character that is of the expected form. The subject sequence shall
64870 contain no characters if the input string is empty or consists entirely of white-space characters,
64871 or if the first non-white-space character is other than a sign or a permissible letter or digit.64872 If the subject sequence has the expected form and the value of *base* is 0, the sequence of
64873 characters starting with the first digit shall be interpreted as an integer constant. If the subject
64874 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the
64875 base for conversion, ascribing to each letter its value as given above. If the subject sequence
64876 begins with a minus-sign, the value resulting from the conversion shall be negated. A pointer to
64877 the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null
64878 pointer.

64879 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
64880 accepted.

64881 If the subject sequence is empty or does not have the expected form, no conversion shall be
64882 performed; the value of *str* shall be stored in the object pointed to by *endptr*, provided that *endptr*
64883 is not a null pointer.

64884 CX The *strtoul()* function shall not change the setting of *errno* if successful.

64885 Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and are also valid returns
64886 on success, an application wishing to check for error situations should set *errno* to 0, then call
64887 *strtoul()* or *strtoull()*, then check *errno*.

64888 RETURN VALUE

64889 Upon successful completion, these functions shall return the converted value, if any. If no
64890 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL]. If the
64891 correct value is outside the range of representable values, {ULONG_MAX} or {ULLONG_MAX}
64892 shall be returned and *errno* set to [ERANGE].

64893 ERRORS

64894 These functions shall fail if:

64895 CX [EINVAL] The value of *base* is not supported.

64896 [ERANGE] The value to be returned is not representable.

64897 These functions may fail if:

64898 CX [EINVAL] No conversion could be performed.

64899 EXAMPLES

64900 None.

64901 APPLICATION USAGE

64902 None.

64903 RATIONALE

64904 None.

64905 FUTURE DIRECTIONS

64906 None.

64907 SEE ALSO

64908 *fscanf()*, *isalpha()*, *strtod()*, *strtol()*

64909 XBD <stdlib.h>

64910 CHANGE HISTORY

64911 First released in Issue 4. Derived from the ANSI C standard.

64912 Issue 5

64913 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

64914 Issue 6

64915 Extensions beyond the ISO C standard are marked.

64916 The following new requirements on POSIX implementations derive from alignment with the
64917 Single UNIX Specification:

- 64918 • The [EINVAL] error condition is added for when the value of *base* is not supported.

strtoul()*System Interfaces*

- 64919
- 64920
- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

64921 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 64922
- 64923
- The *strtoul()* prototype is updated.
 - The *strtoull()* function is added.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***strtoumax()**64924 **NAME**64925 **strtoumax** — convert a string to an integer type64926 **SYNOPSIS**64927 `#include <inttypes.h>`64928 `uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,`
64929 `int base);`64930 **DESCRIPTION**64931 Refer to *strtoimax()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

strxfrm()*System Interfaces*64932 **NAME**64933 `strxfrm, strxfrm_l` — string transformation64934 **SYNOPSIS**64935 `#include <string.h>`

```

64936     size_t strxfrm(char *restrict s1, const char *restrict s2, size_t n);
64937 CX     size_t strxfrm_l(char *restrict s1, const char *restrict s2,
64938         size_t n, locale_t locale);

```

64939 **DESCRIPTION**

64940 CX For `strxfrm()`: The functionality described on this reference page is aligned with the ISO C
 64941 standard. Any conflict between the requirements described here and the ISO C standard is
 64942 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

64943 CX The `strxfrm()` and `strxfrm_l()` functions shall transform the string pointed to by `s2` and place the
 64944 resulting string into the array pointed to by `s1`. The transformation is such that if `strcmp()` is
 64945 applied to two transformed strings, it shall return a value greater than, equal to, or less than 0,
 64946 CX corresponding to the result of `strcoll()` or `strcoll_l()`, respectively, applied to the same two
 64947 CX original strings with the same locale. No more than `n` bytes are placed into the resulting array
 64948 pointed to by `s1`, including the terminating NUL character. If `n` is 0, `s1` is permitted to be a null
 64949 pointer. If copying takes place between objects that overlap, the behavior is undefined.

64950 CX The `strxfrm()` and `strxfrm_l()` functions shall not change the setting of `errno` if successful.

64951 Since no return value is reserved to indicate an error, an application wishing to check for error
 64952 CX situations should set `errno` to 0, then call `strxfrm()` or `strxfrm_l()`, then check `errno`.

64953 **RETURN VALUE**

64954 CX Upon successful completion, `strxfrm()` and `strxfrm_l()` shall return the length of the
 64955 transformed string (not including the terminating NUL character). If the value returned is `n` or
 64956 more, the contents of the array pointed to by `s1` are unspecified.

64957 CX On error, `strxfrm()` and `strxfrm_l()` may set `errno` but no return value is reserved to indicate an
 64958 error.

64959 **ERRORS**

64960 These functions may fail if:

64961 CX [EINVAL] The string pointed to by the `s2` argument contains characters outside the
 64962 domain of the collating sequence.

64963 The `strxfrm_l()` function may fail if:

64964 CX [EINVAL] `locale` is not a valid locale object.

64965 **EXAMPLES**

64966 None.

64967 **APPLICATION USAGE**

64968 The transformation function is such that two transformed strings can be ordered by `strcmp()` as
 64969 appropriate to collating sequence information in the locale of the process (category
 64970 LC_COLLATE).

64971 The fact that when `n` is 0 `s1` is permitted to be a null pointer is useful to determine the size of the
 64972 `s1` array prior to making the transformation.

64973 **RATIONALE**

64974 None.

64975 **FUTURE DIRECTIONS**

64976 None.

64977 **SEE ALSO**64978 *strcmp()*, *strcoll()*

64979 XBD <string.h>

64980 **CHANGE HISTORY**

64981 First released in Issue 3. Included for alignment with the ISO C standard.

64982 **Issue 5**64983 The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.64984 **Issue 6**

64985 Extensions beyond the ISO C standard are marked.

64986 The following new requirements on POSIX implementations derive from alignment with the
64987 Single UNIX Specification:

- 64988 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
64989 added if no conversion could be performed.

64990 The *strxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.64991 **Issue 7**64992 The *strxfrm_l()* function is added from The Open Group Technical Standard, 2006, Extended API
64993 Set Part 4.

swab()

System Interfaces

64994 **NAME**

64995 swab — swap bytes

64996 **SYNOPSIS**

```

64997 XSI    #include <unistd.h>
64998
64998 void swab(const void *restrict src, void *restrict dest,
64999          ssize_t nbytes);

```

65000 **DESCRIPTION**

65001 The *swab()* function shall copy *nbytes* bytes, which are pointed to by *src*, to the object pointed to
 65002 by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd, *swab()*
 65003 copies and exchanges *nbytes*–1 bytes and the disposition of the last byte is unspecified. If
 65004 copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is
 65005 negative, *swab()* does nothing.

65006 **RETURN VALUE**

65007 None.

65008 **ERRORS**

65009 No errors are defined.

65010 **EXAMPLES**

65011 None.

65012 **APPLICATION USAGE**

65013 None.

65014 **RATIONALE**

65015 None.

65016 **FUTURE DIRECTIONS**

65017 None.

65018 **SEE ALSO**65019 XBD [<unistd.h>](#)65020 **CHANGE HISTORY**

65021 First released in Issue 1. Derived from Issue 1 of the SVID.

65022 **Issue 6**

65023 The **restrict** keyword is added to the *swab()* prototype for alignment with the
 65024 ISO/IEC 9899:1999 standard.

*System Interfaces***swprintf()**65025 **NAME**

65026 swprintf — print formatted wide-character output

65027 **SYNOPSIS**

65028 #include <stdio.h>

65029 #include <wchar.h>

65030 int swprintf(wchar_t *restrict ws, size_t n,

65031 const wchar_t *restrict format, ...);

65032 **DESCRIPTION**65033 Refer to *fwprintf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

swscanf()*System Interfaces*65034 **NAME**

65035 swscanf — convert formatted wide-character input

65036 **SYNOPSIS**

65037 #include <stdio.h>

65038 #include <wchar.h>

```
65039 int swscanf(const wchar_t *restrict ws,  
65040             const wchar_t *restrict format, ...);
```

65041 **DESCRIPTION**65042 Refer to *fwscanf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

symlink, symlinkat — make a symbolic link relative to directory file descriptor

SYNOPSIS

```
#include <unistd.h>

int symlink(const char *path1, const char *path2);
int symlinkat(const char *path1, int fd, const char *path2);
```

DESCRIPTION

The *symlink()* function shall create a symbolic link called *path2* that contains the string pointed to by *path1* (*path2* is the name of the symbolic link created, *path1* is the string contained in the symbolic link).

The string pointed to by *path1* shall be treated only as a character string and shall not be validated as a pathname.

If the *symlink()* function fails for any reason other than [EIO], any file named by *path2* shall be unaffected.

The symbolic link's user ID shall be set to the process' effective user ID. The symbolic link's group ID shall be set to the group ID of the parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the symbolic link's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the symbolic link's group ID to the effective group ID of the calling process.

The values of the file mode bits for the created symbolic link are unspecified. All interfaces specified by POSIX.1-2008 shall behave as if the contents of symbolic links can always be read, except that the value of the file mode bits returned in the *st_mode* field of the **stat** structure is unspecified.

Upon successful completion, *symlink()* shall mark for update the last data access, last data modification, and last file status change timestamps of the symbolic link. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

The *symlinkat()* function shall be equivalent to the *symlink()* function except in the case where *path2* specifies a relative path. In this case the symbolic link is created relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function shall not perform the check.

If *symlinkat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working directory is used and the behavior shall be identical to a call to *symlink()*.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return -1 and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

[EACCES]	Write permission is denied in the directory where the symbolic link is being created, or search permission is denied for a component of the path prefix of <i>path2</i> .

symlink()*System Interfaces*

65087	[EEXIST]	The <i>path2</i> argument names an existing file or symbolic link.
65088	[EIO]	An I/O error occurs while reading from or writing to the file system.
65089	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path2</i> argument.
65091	[ENAMETOOLONG]	The length of a component of the pathname specified by the <i>path2</i> argument is longer than {NAME_MAX} or the length of the <i>path1</i> argument is longer than {SYMLINK_MAX}.
65092		
65093		
65094		
65095	[ENOENT]	A component of <i>path2</i> does not name an existing file or <i>path2</i> is an empty string.
65096		
65097	[ENOSPC]	The directory in which the entry for the new symbolic link is being placed cannot be extended because no space is left on the file system containing the directory, or the new symbolic link cannot be created because no space is left on the file system which shall contain the link, or the file system is out of file-allocation resources.
65098		
65099		
65100		
65101		
65102	[ENOTDIR]	A component of the path prefix of <i>path2</i> is not a directory.
65103	[EROFS]	The new symbolic link would reside on a read-only file system.
65104	The <i>symlinkat()</i> function shall fail if:	
65105	[EACCES]	<i>fd</i> was not opened with O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
65106		
65107	[EBADF]	The <i>path2</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
65108		
65109	These functions may fail if:	
65110	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path2</i> argument.
65111		
65112	[ENAMETOOLONG]	The length of the <i>path2</i> argument exceeds {PATH_MAX} or pathname resolution of a symbolic link in the <i>path2</i> argument produced an intermediate result with a length that exceeds {PATH_MAX}.
65113		
65114		
65115		
65116	The <i>symlinkat()</i> function may fail if:	
65117	[ENOTDIR]	The <i>path2</i> argument is not an absolute path and <i>fd</i> is neither AT_FDCWD nor a file descriptor associated with a directory.
65118		

EXAMPLES

None.

APPLICATION USAGE

Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a hard link guarantees the existence of a file, even after the original name has been removed. A symbolic link provides no such assurance; in fact, the file named by the *path1* argument need not exist when the link is created. A symbolic link can cross file system boundaries.

Normal permission checks are made on each component of the symbolic link pathname during its resolution.

RATIONALE

Since POSIX.1-2008 does not require any association of file times with symbolic links, there is no requirement that file times be updated by *symlink()*.

The purpose of the *symlinkat()* function is to create symbolic links in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *symlink()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *symlinkat()* function it can be guaranteed that the created symbolic link is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

fdopendir(), *fstatat()*, *lchown()*, *link()*, *open()*, *readlink()*, *rename()*, *unlink()*

XBD [`<unistd.h>`](#)

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The following changes were made to align with the IEEE P1003.1a draft standard:

- The DESCRIPTION text is updated.
- The [ELOOP] optional error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

The *symlinkat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Additions have been made describing how *symlink()* sets the user and group IDs and file mode of the symbolic link, and its effect on timestamps.

Changes are made to allow a directory to be opened for searching.

sync()

System Interfaces

65156 **NAME**

65157 sync — schedule file system updates

65158 **SYNOPSIS**65159 XSI `#include <unistd.h>`65160 `void sync(void);`65161 **DESCRIPTION**65162 The *sync()* function shall cause all information in memory that updates file systems to be
65163 scheduled for writing out to all file systems.65164 The writing, although scheduled, is not necessarily complete upon return from *sync()*.65165 **RETURN VALUE**65166 The *sync()* function shall not return a value.65167 **ERRORS**

65168 No errors are defined.

65169 **EXAMPLES**

65170 None.

65171 **APPLICATION USAGE**

65172 None.

65173 **RATIONALE**

65174 None.

65175 **FUTURE DIRECTIONS**

65176 None.

65177 **SEE ALSO**65178 *fsync()*65179 XBD `<unistd.h>`65180 **CHANGE HISTORY**

65181 First released in Issue 4, Version 2.

65182 **Issue 5**

65183 Moved from X/OPEN UNIX extension to BASE.

NAME

sysconf — get configurable system variables

SYNOPSIS

#include <unistd.h>

long sysconf(int name);

DESCRIPTION

The *sysconf()* function provides a method for the application to determine the current value of a configurable system limit or option (*variable*). The implementation shall support all of the variables listed in the following table and may support others.

The *name* argument represents the system variable to be queried. The following table lists the minimal set of system variables from <limits.h> or <unistd.h> that can be returned by *sysconf()*, and the symbolic constants defined in <unistd.h> that are the corresponding values used for *name*.

Variable	Value of Name
{AIO_LISTIO_MAX}	_SC_AIO_LISTIO_MAX
{AIO_MAX}	_SC_AIO_MAX
{AIO_PRIO_DELTA_MAX}	_SC_AIO_PRIO_DELTA_MAX
{ARG_MAX}	_SC_ARG_MAX
{ATEXIT_MAX}	_SC_ATEXIT_MAX
{BC_BASE_MAX}	_SC_BC_BASE_MAX
{BC_DIM_MAX}	_SC_BC_DIM_MAX
{BC_SCALE_MAX}	_SC_BC_SCALE_MAX
{BC_STRING_MAX}	_SC_BC_STRING_MAX
{CHILD_MAX}	_SC_CHILD_MAX
Clock ticks/second	_SC_CLK_TCK
{COLL_WEIGHTS_MAX}	_SC_COLL_WEIGHTS_MAX
{DELAYTIMER_MAX}	_SC_DELAYTIMER_MAX
{EXPR_NEST_MAX}	_SC_EXPR_NEST_MAX
{HOST_NAME_MAX}	_SC_HOST_NAME_MAX
{IOV_MAX}	_SC_IOV_MAX
{LINE_MAX}	_SC_LINE_MAX
{LOGIN_NAME_MAX}	_SC_LOGIN_NAME_MAX
{NGROUPS_MAX}	_SC_NGROUPS_MAX
Initial size of <i>getgrgid_r()</i> and <i>getgrnam_r()</i> data buffers	_SC_GETGR_R_SIZE_MAX
Initial size of <i>getpwuid_r()</i> and <i>getpwnam_r()</i> data buffers	_SC_GETPW_R_SIZE_MAX
{MQ_OPEN_MAX}	_SC_MQ_OPEN_MAX
{MQ_PRIO_MAX}	_SC_MQ_PRIO_MAX
{OPEN_MAX}	_SC_OPEN_MAX
_POSIX_ADVISORY_INFO	_SC_ADVISORY_INFO
_POSIX_BARRIERS	_SC_BARRIERS
_POSIX_ASYNCHRONOUS_IO	_SC_ASYNCHRONOUS_IO
_POSIX_CLOCK_SELECTION	_SC_CLOCK_SELECTION
_POSIX_CPUTIME	_SC_CPUTIME
_POSIX_FSYNC	_SC_FSYNC
_POSIX_IPV6	_SC_IPV6
_POSIX_JOB_CONTROL	_SC_JOB_CONTROL

sysconf()*System Interfaces*

	Variable	Value of Name
65232		
65233	_POSIX_MAPPED_FILES	_SC_MAPPED_FILES
65234	_POSIX_MEMLOCK	_SC_MEMLOCK
65235	_POSIX_MEMLOCK_RANGE	_SC_MEMLOCK_RANGE
65236	_POSIX_MEMORY_PROTECTION	_SC_MEMORY_PROTECTION
65237	_POSIX_MESSAGE_PASSING	_SC_MESSAGE_PASSING
65238	_POSIX_MONOTONIC_CLOCK	_SC_MONOTONIC_CLOCK
65239	_POSIX_PRIORITIZED_IO	_SC_PRIORITIZED_IO
65240	_POSIX_PRIORITY_SCHEDULING	_SC_PRIORITY_SCHEDULING
65241	_POSIX_RAW_SOCKETS	_SC_RAW_SOCKETS
65242	_POSIX_READER_WRITER_LOCKS	_SC_READER_WRITER_LOCKS
65243	_POSIX_REALTIME_SIGNALS	_SC_REALTIME_SIGNALS
65244	_POSIX_REGEX	_SC_REGEX
65245	_POSIX_SAVED_IDS	_SC_SAVED_IDS
65246	_POSIX_SEMAPHORES	_SC_SEMAPHORES
65247	_POSIX_SHARED_MEMORY_OBJECTS	_SC_SHARED_MEMORY_OBJECTS
65248	_POSIX_SHELL	_SC_SHELL
65249	_POSIX_SPAWN	_SC_SPAWN
65250	_POSIX_SPIN_LOCKS	_SC_SPIN_LOCKS
65251	_POSIX_SPORADIC_SERVER	_SC_SPORADIC_SERVER
65252	_POSIX_SS_REPL_MAX	_SC_SS_REPL_MAX
65253	_POSIX_SYNCHRONIZED_IO	_SC_SYNCHRONIZED_IO
65254	_POSIX_THREAD_ATTR_STACKADDR	_SC_THREAD_ATTR_STACKADDR
65255	_POSIX_THREAD_ATTR_STACKSIZE	_SC_THREAD_ATTR_STACKSIZE
65256	_POSIX_THREAD_CPU_TIME	_SC_THREAD_CPU_TIME
65257	_POSIX_THREAD_PRIO_INHERIT	_SC_THREAD_PRIO_INHERIT
65258	_POSIX_THREAD_PRIO_PROTECT	_SC_THREAD_PRIO_PROTECT
65259	_POSIX_THREAD_PRIORITY_SCHEDULING	_SC_THREAD_PRIORITY_SCHEDULING
65260	_POSIX_THREAD_PROCESS_SHARED	_SC_THREAD_PROCESS_SHARED
65261	_POSIX_THREAD_ROBUST_PRIO_INHERIT	_SC_THREAD_ROBUST_PRIO_INHERIT
65262	_POSIX_THREAD_ROBUST_PRIO_PROTECT	_SC_THREAD_ROBUST_PRIO_PROTECT
65263	_POSIX_THREAD_SAFE_FUNCTIONS	_SC_THREAD_SAFE_FUNCTIONS
65264	_POSIX_THREAD_SPAORADIC_SERVER	_SC_THREAD_SPAORADIC_SERVER
65265	_POSIX_THREADS	_SC_THREADS
65266	_POSIX_TIMEOUTS	_SC_TIMEOUTS
65267	_POSIX_TIMERS	_SC_TIMERS
65268	_POSIX_TRACE	_SC_TRACE
65269	_POSIX_TRACE_EVENT_FILTER	_SC_TRACE_EVENT_FILTER
65270	_POSIX_TRACE_EVENT_NAME_MAX	_SC_TRACE_EVENT_NAME_MAX
65271	_POSIX_TRACE_INHERIT	_SC_TRACE_INHERIT
65272	_POSIX_TRACE_LOG	_SC_TRACE_LOG
65273	_POSIX_TRACE_NAME_MAX	_SC_TRACE_NAME_MAX
65274	_POSIX_TRACE_SYS_MAX	_SC_TRACE_SYS_MAX
65275	_POSIX_TRACE_USER_EVENT_MAX	_SC_TRACE_USER_EVENT_MAX
65276	_POSIX_TYPED_MEMORY_OBJECTS	_SC_TYPED_MEMORY_OBJECTS
65277	_POSIX_VERSION	_SC_VERSION
65278	_POSIX_V7_ILP32_OFF32	_SC_V7_ILP32_OFF32
65279	_POSIX_V7_ILP32_OFFBIG	_SC_V7_ILP32_OFFBIG
65280	_POSIX_V7_LP64_OFF64	_SC_V7_LP64_OFF64
65281	_POSIX_V7_LPBIG_OFFBIG	_SC_V7_LPBIG_OFFBIG

	Variable	Value of Name
65282		
65283	_POSIX_V6_ILP32_OFF32	_SC_V6_ILP32_OFF32
65284	_POSIX_V6_ILP32_OFFBIG	_SC_V6_ILP32_OFFBIG
65285	_POSIX_V6_LP64_OFF64	_SC_V6_LP64_OFF64
65286	_POSIX_V6_LPBIG_OFFBIG	_SC_V6_LPBIG_OFFBIG
65287	_POSIX2_C_BIND	_SC_2_C_BIND
65288	_POSIX2_C_DEV	_SC_2_C_DEV
65289	_POSIX2_CHAR_TERM	_SC_2_CHAR_TERM
65290	_POSIX2_FORT_DEV	_SC_2_FORT_DEV
65291	_POSIX2_FORT_RUN	_SC_2_FORT_RUN
65292	_POSIX2_LOCALEDEF	_SC_2_LOCALEDEF
65293	_POSIX2_PBS	_SC_2_PBS
65294	_POSIX2_PBS_ACCOUNTING	_SC_2_PBS_ACCOUNTING
65295	_POSIX2_PBS_CHECKPOINT	_SC_2_PBS_CHECKPOINT
65296	_POSIX2_PBS_LOCATE	_SC_2_PBS_LOCATE
65297	_POSIX2_PBS_MESSAGE	_SC_2_PBS_MESSAGE
65298	_POSIX2_PBS_TRACK	_SC_2_PBS_TRACK
65299	_POSIX2_SW_DEV	_SC_2_SW_DEV
65300	_POSIX2_UPE	_SC_2_UPE
65301	_POSIX2_VERSION	_SC_2_VERSION
65302	{PAGE_SIZE}	_SC_PAGE_SIZE
65303	{PAGESIZE}	_SC_PAGESIZE
65304	{PTHREAD_DESTRUCTOR_ITERATIONS}	_SC_THREAD_DESTRUCTOR_ITERATIONS
65305	{PTHREAD_KEYS_MAX}	_SC_THREAD_KEYS_MAX
65306	{PTHREAD_STACK_MIN}	_SC_THREAD_STACK_MIN
65307	{PTHREAD_THREADS_MAX}	_SC_THREAD_THREADS_MAX
65308	{RE_DUP_MAX}	_SC_RE_DUP_MAX
65309	{RTSIG_MAX}	_SC_RTSIG_MAX
65310	{SEM_NSEMS_MAX}	_SC_SEM_NSEMS_MAX
65311	{SEM_VALUE_MAX}	_SC_SEM_VALUE_MAX
65312	{SIGQUEUE_MAX}	_SC_SIGQUEUE_MAX
65313	{STREAM_MAX}	_SC_STREAM_MAX
65314	{SYMLOOP_MAX}	_SC_SYMLOOP_MAX
65315	{TIMER_MAX}	_SC_TIMER_MAX
65316	{TTY_NAME_MAX}	_SC_TTY_NAME_MAX
65317	{TZNAME_MAX}	_SC_TZNAME_MAX
65318	_XOPEN_CRYPT	_SC_XOPEN_CRYPT
65319	_XOPEN_ENH_I18N	_SC_XOPEN_ENH_I18N
65320	_XOPEN_REALTIME	_SC_XOPEN_REALTIME
65321	_XOPEN_REALTIME_THREADS	_SC_XOPEN_REALTIME_THREADS
65322	_XOPEN_SHM	_SC_XOPEN_SHM
65323	_XOPEN_STREAMS	_SC_XOPEN_STREAMS
65324	_XOPEN_UNIX	_SC_XOPEN_UNIX
65325	_XOPEN_UUCP	_SC_XOPEN_UUCP
65326	_XOPEN_VERSION	_SC_XOPEN_VERSION

RETURN VALUE

If *name* is an invalid value, *sysconf()* shall return -1 and set *errno* to indicate the error. If the variable corresponding to *name* is described in **<limits.h>** as a maximum or minimum value and the variable has no limit, *sysconf()* shall return -1 without changing the value of *errno*. Note that indefinite limits do not imply infinite limits; see **<limits.h>**.

Otherwise, *sysconf()* shall return the current variable value on the system. The value returned

sysconf()*System Interfaces*

65333 shall not be more restrictive than the corresponding value described to the application when it
 65334 was compiled with the implementation's **<limits.h>** or **<unistd.h>**. The value shall not change
 65335 XSI during the lifetime of the calling process, except that *sysconf(_SC_OPEN_MAX)* may return
 65336 different values before and after a call to *setrlimit()* which changes the RLIMIT_NOFILE soft
 65337 limit.

65338 If the variable corresponding to *name* is dependent on an unsupported option, the results are
 65339 unspecified.

ERRORS

65341 The *sysconf()* function shall fail if:

65342 [EINVAL] The value of the *name* argument is invalid.

EXAMPLES

65343 None.
 65344

APPLICATION USAGE

65346 As -1 is a permissible return value in a successful situation, an application wishing to check for
 65347 error situations should set *errno* to 0, then call *sysconf()*, and, if it returns -1, check to see if *errno*
 65348 is non-zero.

65349 Application developers should check whether an option, such as *_POSIX_TRACE*, is supported
 65350 prior to obtaining and using values for related variables such as *_POSIX_TRACE_NAME_MAX*.

RATIONALE

65352 This functionality was added in response to requirements of application developers and of
 65353 system vendors who deal with many international system configurations. It is closely related to
 65354 *pathconf()* and *fpathconf()*.

65355 Although a conforming application can run on all systems by never demanding more resources
 65356 than the minimum values published in this volume of POSIX.1-2008, it is useful for that
 65357 application to be able to use the actual value for the quantity of a resource available on any
 65358 given system. To do this, the application makes use of the value of a symbolic constant in
 65359 **<limits.h>** or **<unistd.h>**.

65360 However, once compiled, the application must still be able to cope if the amount of resource
 65361 available is increased. To that end, an application may need a means of determining the quantity
 65362 of a resource, or the presence of an option, at execution time.

65363 Two examples are offered:

- 65364 1. Applications may wish to act differently on systems with or without job control.
 65365 Applications vendors who wish to distribute only a single binary package to all instances
 65366 of a computer architecture would be forced to assume job control is never available if it
 65367 were to rely solely on the **<unistd.h>** value published in this volume of POSIX.1-2008.
- 65368 2. International applications vendors occasionally require knowledge of the number of clock
 65369 ticks per second. Without these facilities, they would be required to either distribute their
 65370 applications partially in source form or to have 50 Hz and 60 Hz versions for the various
 65371 countries in which they operate.

65372 It is the knowledge that many applications are actually distributed widely in executable form
 65373 that leads to this facility. If limited to the most restrictive values in the headers, such applications
 65374 would have to be prepared to accept the most limited environments offered by the smallest
 65375 microcomputers. Although this is entirely portable, there was a consensus that they should be
 65376 able to take advantage of the facilities offered by large systems, without the restrictions
 65377 associated with source and object distributions.

During the discussions of this feature, it was pointed out that it is almost always possible for an application to discern what a value might be at runtime by suitably testing the various functions themselves. And, in any event, it could always be written to adequately deal with error returns from the various functions. In the end, it was felt that this imposed an unreasonable level of complication and sophistication on the application developer.

This runtime facility is not meant to provide ever-changing values that applications have to check multiple times. The values are seen as changing no more frequently than once per system initialization, such as by a system administrator or operator with an automatic configuration program. This volume of POSIX.1-2008 specifies that they shall not change within the lifetime of the process.

Some values apply to the system overall and others vary at the file system or directory level. The latter are described in *fpathconf()*.

Note that all values returned must be expressible as integers. String values were considered, but the additional flexibility of this approach was rejected due to its added complexity of implementation and use.

Some values, such as {PATH_MAX}, are sometimes so large that they must not be used to, say, allocate arrays. The *sysconf()* function returns a negative value to show that this symbolic constant is not even defined in this case.

Similar to *pathconf()*, this permits the implementation not to have a limit. When one resource is infinite, returning an error indicating that some other resource limit has been reached is conforming behavior.

FUTURE DIRECTIONS

None.

SEE ALSO

confstr(), *fpathconf()*

XBD *<limits.h>*, *<unistd.h>*

XCU *getconf*

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

The XBS_ variables and name values are added to the table of system variables in the DESCRIPTION. These are all marked EX.

Issue 6

The symbol CLK_TCK is obsolescent and removed. It is replaced with the phrase “clock ticks per second”.

The symbol {PASS_MAX} is removed.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Table entries are added for the following variables: *_SC_REGEX*, *_SC_SHELL*, *_SC_REGEX_VERSION*, *_SC_SYMLINK_MAX*.

The following *sysconf()* variables and their associated names are added for alignment with IEEE Std 1003.1d-1999:

sysconf()*System Interfaces*

65421 _POSIX_ADVISORY_INFO
 65422 _POSIX_CPUTIME
 65423 _POSIX_SPAWN
 65424 _POSIX_SPORADIC_SERVER
 65425 _POSIX_THREAD_CPUTIME
 65426 _POSIX_THREAD_SPORADIC_SERVER
 65427 _POSIX_TIMEOUTS

65428 The following changes are made to the DESCRIPTION for alignment with IEEE Std 1003.1-2000:

- 65429 • A statement expressing the dependency of support for some system variables on
- 65430 implementation options is added.
- 65431 • The following system variables are added:

65432 _POSIX_BARRIERS
 65433 _POSIX_CLOCK_SELECTION
 65434 _POSIX_MONOTONIC_CLOCK
 65435 _POSIX_READER_WRITER_LOCKS
 65436 _POSIX_SPIN_LOCKS
 65437 _POSIX_TYPED_MEMORY_OBJECTS

65438 The following system variables are added for alignment with IEEE Std 1003.2d-1994:

65439 _POSIX2_PBS
 65440 _POSIX2_PBS_ACCOUNTING
 65441 _POSIX2_PBS_LOCATE
 65442 _POSIX2_PBS_MESSAGE
 65443 _POSIX2_PBS_TRACK

65444 The following *sysconf()* variables and their associated names are added for alignment with

65445 IEEE Std 1003.1q-2000:

65446 _POSIX_TRACE
 65447 _POSIX_TRACE_EVENT_FILTER
 65448 _POSIX_TRACE_INHERIT
 65449 _POSIX_TRACE_LOG

65450 The macros associated with the *c89* programming models are marked LEGACY, and new

65451 equivalent macros associated with *c99* are introduced.

65452 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/62 is applied, updating the

65453 DESCRIPTION to denote that the *_PC** and *_SC** symbols are now required to be supported. A

65454 corresponding change has been made in the Base Definitions volume of POSIX.1-2008. The

65455 deletion in the second paragraph removes some duplicated text. Additional symbols that were

65456 erroneously omitted from this reference page have been added.

65457 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/63 is applied, making it clear in the

65458 RETURN VALUE section that the value returned for *sysconf(_SC_OPEN_MAX)* may change if a

65459 call to *setrlimit()* adjusts the RLIMIT_NOFILE soft limit.

65460 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/134 is applied, updating the

65461 DESCRIPTION to remove an erroneous entry for *_POSIX_SYMLOOP_MAX*. This corrects an

65462 error in IEEE Std 1003.1-2001/Cor 1-2002.

65463 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/135 is applied, removing

65464 _POSIX_FILE_LOCKING, _POSIX_MULTI_PROCESS, _POSIX2_C_VERSION, and
65465 _XOPEN_XCU_VERSION (and their associated _SC_* variables) from the DESCRIPTION and
65466 APPLICATION USAGE sections.

65467 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/136 is applied, adding the following
65468 constants (and their associated _SC_* variables) to the DESCRIPTION:

65469 _POSIX_SS_REPL_MAX
65470 _POSIX_TRACE_EVENT_NAME_MAX
65471 _POSIX_TRACE_NAME_MAX
65472 _POSIX_TRACE_SYS_MAX
65473 _POSIX_TRACE_USER_EVENT_MAX

65474 The RETURN VALUE and APPLICATION USAGE sections are updated to note that if variables
65475 are dependent on unsupported options, the results are unspecified.

65476 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/137 is applied, removing
65477 _REGEX_VERSION and _SC_REGEX_VERSION.

65478 **Issue 7**

65479 Austin Group Interpretation 1003.1-2001 #160 is applied.

65480 SD5-XSH-ERN-166 is applied, changing “Maximum size” to “Initial size” for the “Maximum
65481 size of ...” entries in the table in the DESCRIPTION.

65482 The variables for the supported programming environments are updated to be V7 and the
65483 LEGACY variables are removed.

65484 The following constants are added:

65485 _POSIX_THREAD_ROBUST_PRIO_INHERIT
65486 _POSIX_THREAD_ROBUST_PRIO_PROTECT

65487 The _XOPEN_UUCP variable and its associated _SC_XOPEN_UUCP value is added to the table
65488 of system variables.

syslog()*System Interfaces*65489 **NAME**

65490 syslog — log a message

65491 **SYNOPSIS**65492 XSI `#include <syslog.h>`65493 `void syslog(int priority, const char *message, ... /* argument */);`65494 **DESCRIPTION**65495 Refer to *closelog()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

system()

NAME

system — issue a command

SYNOPSIS

```
#include <stdlib.h>
```

```
int system(const char *command);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

If *command* is a null pointer, the *system()* function shall determine whether the host environment has a command processor. If *command* is not a null pointer, the *system()* function shall pass the string pointed to by *command* to that command processor to be executed in an implementation-defined manner; this might then cause the program calling *system()* to behave in a non-conforming manner or to terminate.

The *system()* function shall behave as if a child process were created using *fork()*, and the child process invoked the *sh* utility using *execl()* as follows:

```
execl(<shell path>, "sh", "-c", command, (char *)0);
```

where *<shell path>* is an unspecified pathname for the *sh* utility. It is unspecified whether the handlers registered with *pthread_atfork()* are called as part of the creation of the child process.

The *system()* function shall ignore the SIGINT and SIGQUIT signals, and shall block the SIGCHLD signal, while waiting for the command to terminate. If this might cause the application to miss a signal that would have killed it, then the application should examine the return value from *system()* and take whatever action is appropriate to the application if the command terminated due to receipt of a signal.

The *system()* function shall not affect the termination status of any child of the calling processes other than the process or processes it itself creates.

The *system()* function shall not return until the child process has terminated.

The *system()* function need not be thread-safe.

RETURN VALUE

If *command* is a null pointer, *system()* shall return non-zero to indicate that a command processor is available, or zero if none is available. The *system()* function shall always return non-zero when *command* is NULL.

If *command* is not a null pointer, *system()* shall return the termination status of the command language interpreter in the format specified by *waitpid()*. The termination status shall be as defined for the *sh* utility; otherwise, the termination status is unspecified. If some error prevents the command language interpreter from executing after the child process is created, the return value from *system()* shall be as if the command language interpreter had terminated using *exit(127)* or *_exit(127)*. If a child process cannot be created, or if the termination status for the command language interpreter cannot be obtained, *system()* shall return -1 and set *errno* to indicate the error.

ERRORS

The *system()* function may set *errno* values as described by *fork()*.

system()*System Interfaces*

65538 In addition, *system()* may fail if:

65539 CX [ECHILD] The status of the child process created by *system()* is no longer available.

EXAMPLES

65541 None.

APPLICATION USAGE

65543 If the return value of *system()* is not -1 , its value can be decoded through the use of the macros
65544 described in `<sys/wait.h>`. For convenience, these macros are also provided in `<stdlib.h>`.

65545 Note that, while *system()* must ignore SIGINT and SIGQUIT and block SIGCHLD while waiting
65546 for the child to terminate, the handling of signals in the executed command is as specified by
65547 *fork()* and *exec*. For example, if SIGINT is being caught or is set to SIG_DFL when *system()* is
65548 called, then the child is started with SIGINT handling set to SIG_DFL.

65549 Ignoring SIGINT and SIGQUIT in the parent process prevents coordination problems (two
65550 processes reading from the same terminal, for example) when the executed command ignores or
65551 catches one of the signals. It is also usually the correct action when the user has given a
65552 command to the application to be executed synchronously (as in the '!' command in many
65553 interactive applications). In either case, the signal should be delivered only to the child process,
65554 not to the application itself. There is one situation where ignoring the signals might have less
65555 than the desired effect. This is when the application uses *system()* to perform some task invisible
65556 to the user. If the user typed the interrupt character ("^C", for example) while *system()* is being
65557 used in this way, one would expect the application to be killed, but only the executed command
65558 is killed. Applications that use *system()* in this way should carefully check the return status from
65559 *system()* to see if the executed command was successful, and should take appropriate action
65560 when the command fails.

65561 Blocking SIGCHLD while waiting for the child to terminate prevents the application from
65562 catching the signal and obtaining status from *system()*'s child process before *system()* can get the
65563 status itself.

65564 The context in which the utility is ultimately executed may differ from that in which *system()*
65565 was called. For example, file descriptors that have the FD_CLOEXEC flag set are closed, and the
65566 process ID and parent process ID are different. Also, if the executed utility changes its
65567 environment variables or its current working directory, that change is not reflected in the caller's
65568 context.

65569 There is no defined way for an application to find the specific path for the shell. However,
65570 *confstr()* can provide a value for *PATH* that is guaranteed to find the *sh* utility.

65571 Using the *system()* function in more than one thread in a process or when the SIGCHLD signal is
65572 being manipulated by more than one thread in a process may produce unexpected results.

RATIONALE

65573 The *system()* function should not be used by programs that have set user (or group) ID
65574 privileges. The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used
65575 instead. This prevents any unforeseen manipulation of the environment of the user that could
65576 cause execution of commands not anticipated by the calling program.
65577

65578 There are three levels of specification for the *system()* function. The ISO C standard gives the
65579 most basic. It requires that the function exists, and defines a way for an application to query
65580 whether a command language interpreter exists. It says nothing about the command language
65581 or the environment in which the command is interpreted.

65582 POSIX.1-2008 places additional restrictions on *system()*. It requires that if there is a command
65583 language interpreter, the environment must be as specified by *fork()* and *exec*. This ensures, for

example, that *close-on-exec* works, that file locks are not inherited, and that the process ID is different. It also specifies the return value from *system()* when the command line can be run, thus giving the application some information about the command's completion status.

Finally, POSIX.1-2008 requires the command to be interpreted as in the shell command language defined in the Shell and Utilities volume of POSIX.1-2008.

Note that, *system(NULL)* is required to return non-zero, indicating that there is a command language interpreter. At first glance, this would seem to conflict with the ISO C standard which allows *system(NULL)* to return zero. There is no conflict, however. A system must have a command language interpreter, and is non-conforming if none is present. It is therefore permissible for the *system()* function on such a system to implement the behavior specified by the ISO C standard as long as it is understood that the implementation does not conform to POSIX.1-2008 if *system(NULL)* returns zero.

It was explicitly decided that when *command* is NULL, *system()* should not be required to check to make sure that the command language interpreter actually exists with the correct mode, that there are enough processes to execute it, and so on. The call *system(NULL)* could, theoretically, check for such problems as too many existing child processes, and return zero. However, it would be inappropriate to return zero due to such a (presumably) transient condition. If some condition exists that is not under the control of this application and that would cause any *system()* call to fail, that system has been rendered non-conforming.

Early drafts required, or allowed, *system()* to return with *errno* set to [EINTR] if it was interrupted with a signal. This error return was removed, and a requirement that *system()* not return until the child has terminated was added. This means that if a *waitpid()* call in *system()* exits with *errno* set to [EINTR], *system()* must reissue the *waitpid()*. This change was made for two reasons:

1. There is no way for an application to clean up if *system()* returns [EINTR], short of calling *wait()*, and that could have the undesirable effect of returning the status of children other than the one started by *system()*.
2. While it might require a change in some historical implementations, those implementations already have to be changed because they use *wait()* instead of *waitpid()*.

Note that if the application is catching SIGCHLD signals, it will receive such a signal before a successful *system()* call returns.

To conform to POSIX.1-2008, *system()* must use *waitpid()*, or some similar function, instead of *wait()*.

The following code sample illustrates how *system()* might be implemented on an implementation conforming to POSIX.1-2008.

```
#include <signal.h>
int system(const char *cmd)
{
    int stat;
    pid_t pid;
    struct sigaction sa, savintr, savequit;
    sigset_t saveblock;
    if (cmd == NULL)
        return(1);
    sa.sa_handler = SIG_IGN;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
```

system()*System Interfaces*

```

65631     sigemptyset(&savintr.sa_mask);
65632     sigemptyset(&savequit.sa_mask);
65633     sigaction(SIGINT, &sa, &savintr);
65634     sigaction(SIGQUIT, &sa, &savequit);
65635     sigaddset(&sa.sa_mask, SIGCHLD);
65636     sigprocmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
65637     if ((pid = fork()) == 0) {
65638         sigaction(SIGINT, &savintr, (struct sigaction *)0);
65639         sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
65640         sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
65641         execl("/bin/sh", "sh", "-c", cmd, (char *)0);
65642         _exit(127);
65643     }
65644     if (pid == -1) {
65645         stat = -1; /* errno comes from fork() */
65646     } else {
65647         while (waitpid(pid, &stat, 0) == -1) {
65648             if (errno != EINTR) {
65649                 stat = -1;
65650                 break;
65651             }
65652         }
65653     }
65654     sigaction(SIGINT, &savintr, (struct sigaction *)0);
65655     sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
65656     sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
65657     return(stat);
65658 }

```

Note that, while a particular implementation of *system()* (such as the one above) can assume a particular path for the shell, such a path is not necessarily valid on another system. The above example is not portable, and is not intended to be.

One reviewer suggested that an implementation of *system()* might want to use an environment variable such as *SHELL* to determine which command interpreter to use. The supposed implementation would use the default command interpreter if the one specified by the environment variable was not available. This would allow a user, when using an application that prompts for command lines to be processed using *system()*, to specify a different command interpreter. Such an implementation is discouraged. If the alternate command interpreter did not follow the command line syntax specified in the Shell and Utilities volume of POSIX.1-2008, then changing *SHELL* would render *system()* non-conforming. This would affect applications that expected the specified behavior from *system()*, and since the Shell and Utilities volume of POSIX.1-2008 does not mention that *SHELL* affects *system()*, the application would not know that it needed to unset *SHELL*.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *pipe()*, *pthread_atfork()*, *wait()*

XBD *<limits.h>*, *<signal.h>*, *<stdlib.h>*, *<sys/wait.h>*

XCU *sh*

65679 **CHANGE HISTORY**

65680 First released in Issue 1. Derived from Issue 1 of the SVID.

65681 **Issue 6**

65682 Extensions beyond the ISO C standard are marked.

65683 **Issue 7**65684 Austin Group Interpretation 1003.1-2001 #055 is applied, clarifying the thread-safety of this
65685 function and treatment of *at_fork()* handlers.

65686 Austin Group Interpretation 1003.1-2001 #156 is applied.

65687 SD5-XSH-ERN-30 is applied.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

tan()

System Interfaces

65688 **NAME**65689 **tan, tanf, tanl** — tangent function65690 **SYNOPSIS**

```

65691 #include <math.h>
65692 double tan(double x);
65693 float tanf(float x);
65694 long double tanl(long double x);

```

65695 **DESCRIPTION**

65696 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 65697 conflict between the requirements described here and the ISO C standard is unintentional. This
 65698 volume of POSIX.1-2008 defers to the ISO C standard.

65699 These functions shall compute the tangent of their argument *x*, measured in radians.

65700 An application wishing to check for error situations should set *errno* to zero and call
 65701 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 65702 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 65703 zero, an error has occurred.

65704 **RETURN VALUE**

65705 Upon successful completion, these functions shall return the tangent of *x*.

65706 If the correct value would cause underflow, and is not representable, a range error may occur,
 65707 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

65708 **MX** If *x* is NaN, a NaN shall be returned.

65709 If *x* is ± 0 , *x* shall be returned.

65710 If *x* is subnormal, a range error may occur and *x* should be returned.

65711 If *x* is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 65712 defined value shall be returned.

65713 If the correct value would cause underflow, and is representable, a range error may occur and
 65714 the correct value shall be returned.

65715 **XSI** If the correct value would cause overflow, a range error shall occur and *tan()*, *tanf()*, and *tanl()*
 65716 shall return $\pm \text{HUGE\_VAL}$, $\pm \text{HUGE\_VALF}$, and $\pm \text{HUGE\_VALL}$, respectively, with the same sign
 65717 as the correct value of the function.

65718 **ERRORS**

65719 These functions shall fail if:

65720 **MX** **Domain Error** The value of *x* is $\pm \text{Inf}$.

65721 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 65722 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 65723 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 65724 shall be raised.

65725 **XSI** **Range Error** The result overflows

65726 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 65727 then *errno* shall be set to [ERANGE]. If the integer expression
 65728 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 65729 floating-point exception shall be raised.

65730 These functions may fail if:

65731 MX Range Error The result underflows, or the value of x is subnormal.

65732 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 65733 then *errno* shall be set to [ERANGE]. If the integer expression
 65734 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 65735 floating-point exception shall be raised.

65736 EXAMPLES

65737 Taking the Tangent of a 45-Degree Angle

```
65738 #include <math.h>
65739 ...
65740 double radians = 45.0 * M_PI / 180;
65741 double result;
65742 ...
65743 result = tan (radians);
```

65744 APPLICATION USAGE

65745 There are no known floating-point representations such that for a normal argument, $\tan(x)$ is
 65746 either overflow or underflow.

65747 These functions may lose accuracy when their argument is near a multiple of $\pi/2$ or is far from
 65748 0.0.

65749 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 65750 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

65751 RATIONALE

65752 None.

65753 FUTURE DIRECTIONS

65754 None.

65755 SEE ALSO

65756 [atan\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)

65757 XBD Section 4.19 (on page 116), [<math.h>](#)

65758 CHANGE HISTORY

65759 First released in Issue 1. Derived from Issue 1 of the SVID.

65760 Issue 5

65761 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes
 65762 in previous issues.

65763 Issue 6

65764 The *tanf()* and *tanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

65765 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 65766 revised to align with the ISO/IEC 9899:1999 standard.

65767 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 65768 marked.

65769 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/64 is applied, correcting the last
 65770 paragraph in the RETURN VALUE section.

tanh()

System Interfaces

65771 **NAME**

65772 tanh, tanhf, tanhl — hyperbolic tangent functions

65773 **SYNOPSIS**

65774 #include <math.h>

65775 double tanh(double x);

65776 float tanhf(float x);

65777 long double tanhl(long double x);

65778 **DESCRIPTION**

65779 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 65780 conflict between the requirements described here and the ISO C standard is unintentional. This
 65781 volume of POSIX.1-2008 defers to the ISO C standard.

65782 These functions shall compute the hyperbolic tangent of their argument *x*.

65783 An application wishing to check for error situations should set *errno* to zero and call
 65784 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 65785 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 65786 zero, an error has occurred.

65787 **RETURN VALUE**65788 Upon successful completion, these functions shall return the hyperbolic tangent of *x*.65789 **MX** If *x* is NaN, a NaN shall be returned.65790 If *x* is ± 0 , *x* shall be returned.65791 If *x* is $\pm \text{Inf}$, ± 1 shall be returned.65792 If *x* is subnormal, a range error may occur and *x* should be returned.65793 **ERRORS**

65794 These functions may fail if:

65795 **MX** **Range Error** The value of *x* is subnormal.

65796 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 65797 then *errno* shall be set to [ERANGE]. If the integer expression
 65798 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 65799 floating-point exception shall be raised.

65800 **EXAMPLES**

65801 None.

65802 **APPLICATION USAGE**

65803 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 65804 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

65805 **RATIONALE**

65806 None.

65807 **FUTURE DIRECTIONS**

65808 None.

65809 **SEE ALSO**65810 *atanh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*

65811 XBD Section 4.19 (on page 116), <math.h>

65812 **CHANGE HISTORY**

65813 First released in Issue 1. Derived from Issue 1 of the SVID.

65814 **Issue 5**65815 The DESCRIPTION is updated to indicate how an application should check for an error. This
65816 text was previously published in the APPLICATION USAGE section.65817 **Issue 6**65818 The *tanhf()* and *tanhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.65819 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
65820 revised to align with the ISO/IEC 9899:1999 standard.65821 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
65822 marked.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

tanl()*System Interfaces*65823 **NAME**

65824 tanl — tangent function

65825 **SYNOPSIS**

65826 #include <math.h>

65827 long double tanl(long double x);

65828 **DESCRIPTION**65829 Refer to *tan()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

tcdrain — wait for transmission of output

SYNOPSIS

```
#include <termios.h>

int tcdrain(int fildes);
```

DESCRIPTION

The *tcdrain()* function shall block until all output written to the object referred to by *fildes* is transmitted. The *fildes* argument is an open file descriptor associated with a terminal.

Any attempts to use *tcdrain()* from a process which is a member of a background process group on a *fildes* associated with its controlling terminal, shall cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcdrain()* function shall fail if:

- [EBADF] The *fildes* argument is not a valid file descriptor.
- [EINTR] A signal interrupted *tcdrain()*.
- [ENOTTY] The file associated with *fildes* is not a terminal.

The *tcdrain()* function may fail if:

- [EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[tcflush\(\)](#)
[XBD Chapter 11](#) (on page 199), [<termios.h>](#)

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

tcdrain()*System Interfaces*65866 **Issue 6**

65867 The following new requirements on POSIX implementations derive from alignment with the
65868 Single UNIX Specification:

- 65869 • In the DESCRIPTION, the final paragraph is no longer conditional on
65870 _POSIX_JOB_CONTROL. This is a FIPS requirement.
- 65871 • The [EIO] error is added.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

tcflow — suspend or restart the transmission or reception of data

SYNOPSIS

```
#include <termios.h>

int tcflow(int fildes, int action);
```

DESCRIPTION

The *tcflow()* function shall suspend or restart transmission or reception of data on the object referred to by *fildes*, depending on the value of *action*. The *fildes* argument is an open file descriptor associated with a terminal.

- If *action* is TCOOFF, output shall be suspended.
- If *action* is TCOON, suspended output shall be restarted.
- If *action* is TCIOFF and *fildes* refers to a terminal device, the system shall transmit a STOP character, which is intended to cause the terminal device to stop transmitting data to the system. If *fildes* is associated with a pseudo-terminal, the STOP character need not be transmitted.
- If *action* is TCION and *fildes* refers to a terminal device, the system shall transmit a START character, which is intended to cause the terminal device to start transmitting data to the system. If *fildes* is associated with a pseudo-terminal, the START character need not be transmitted.

The default on the opening of a terminal file is that neither its input nor its output are suspended.

Attempts to use *tcflow()* from a process which is a member of a background process group on a *fildes* associated with its controlling terminal, shall cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcflow()* function shall fail if:

- [EBADF] The *fildes* argument is not a valid file descriptor.
- [EINVAL] The *action* argument is not a supported value.
- [ENOTTY] The file associated with *fildes* is not a terminal.

The *tcflow()* function may fail if:

- [EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

tcflow()*System Interfaces*65908 **EXAMPLES**

65909 None.

65910 **APPLICATION USAGE**

65911 None.

65912 **RATIONALE**

65913 None.

65914 **FUTURE DIRECTIONS**

65915 None.

65916 **SEE ALSO**65917 [*tcsendbreak\(\)*](#)65918 XBD [Chapter 11](#) (on page 199), [**<termios.h>**](#)65919 **CHANGE HISTORY**

65920 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

65921 **Issue 6**65922 The following new requirements on POSIX implementations derive from alignment with the
65923 Single UNIX Specification:

- 65924
- The [EIO] error is added.

65925 **Issue 7**65926 SD5-XSH-ERN-190 is applied, clarifying in the DESCRIPTION the transmission of START and
65927 STOP characters.

NAME

tcflush — flush non-transmitted output data, non-read input data, or both

SYNOPSIS

```
#include <termios.h>
```

```
int tcflush(int fildes, int queue_selector);
```

DESCRIPTION

Upon successful completion, *tcflush()* shall discard data written to the object referred to by *fildes* (an open file descriptor associated with a terminal) but not transmitted, or data received but not read, depending on the value of *queue_selector*:

- If *queue_selector* is TCIFLUSH, it shall flush data received but not read.
- If *queue_selector* is TCOFLUSH, it shall flush data written but not transmitted.
- If *queue_selector* is TCIOFLUSH, it shall flush both data received but not read and data written but not transmitted.

Attempts to use *tcflush()* from a process which is a member of a background process group on a *fildes* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcflush()* function shall fail if:

[EBADF]	The <i>fildes</i> argument is not a valid file descriptor.
[EINVAL]	The <i>queue_selector</i> argument is not a supported value.
[ENOTTY]	The file associated with <i>fildes</i> is not a terminal.

The *tcflush()* function may fail if:

[EIO]	The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[*tcdrain\(\)*](#)

XBD Chapter 11 (on page 199), [*<termios.h>*](#)

tcflush()*System Interfaces*65967 **CHANGE HISTORY**

65968 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

65969 **Issue 6**65970 The Open Group Corrigendum U035/1 is applied. In the ERRORS and APPLICATION USAGE
65971 sections, references to *tcflow()* are replaced with *tcflush()*.65972 The following new requirements on POSIX implementations derive from alignment with the
65973 Single UNIX Specification:

- 65974 • In the DESCRIPTION, the final paragraph is no longer conditional on
65975 `_POSIX_JOB_CONTROL`. This is a FIPS requirement.
- 65976 • The [EIO] error is added.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

tcgetattr()**NAME**

tcgetattr — get the parameters associated with the terminal

SYNOPSIS

```
#include <termios.h>
```

```
int tcgetattr(int fildes, struct termios *termios_p);
```

DESCRIPTION

The **tcgetattr()** function shall get the parameters associated with the terminal referred to by *fildes* and store them in the **termios** structure referenced by *termios_p*. The *fildes* argument is an open file descriptor associated with a terminal.

The *termios_p* argument is a pointer to a **termios** structure.

The **tcgetattr()** operation is allowed from any process.

If the terminal device supports different input and output baud rates, the baud rates stored in the **termios** structure returned by **tcgetattr()** shall reflect the actual baud rates, even if they are equal. If differing baud rates are not supported, the rate returned as the output baud rate shall be the actual baud rate. If the terminal device does not support split baud rates, the input baud rate stored in the **termios** structure shall be the output rate (as one of the symbolic values).

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The **tcgetattr()** function shall fail if:

[EBADF]	The <i>fildes</i> argument is not a valid file descriptor.
[ENOTTY]	The file associated with <i>fildes</i> is not a terminal.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Care must be taken when changing the terminal attributes. Applications should always do a **tcgetattr()**, save the **termios** structure values returned, and then do a **tcsetattr()**, changing only the necessary fields. The application should use the values saved from the **tcgetattr()** to reset the terminal state whenever it is done with the terminal. This is necessary because terminal attributes apply to the underlying port and not to each individual open instance; that is, all processes that have used the terminal see the latest attribute changes.

A program that uses these functions should be written to catch all signals and take other appropriate actions to ensure that when the program terminates, whether planned or not, the terminal device's state is restored to its original state.

Existing practice dealing with error returns when only part of a request can be honored is based on calls to the **ioctl()** function. In historical BSD and System V implementations, the corresponding **ioctl()** returns zero if the requested actions were semantically correct, even if some of the requested changes could not be made. Many existing applications assume this behavior and would no longer work correctly if the return value were changed from zero to -1 in this case.

Note that either specification has a problem. When zero is returned, it implies everything

tcgetattr()*System Interfaces*

66021 succeeded even if some of the changes were not made. When -1 is returned, it implies
66022 everything failed even though some of the changes were made.

66023 Applications that need all of the requested changes made to work properly should follow
66024 *tcsetattr()* with a call to *tcgetattr()* and compare the appropriate field values.

66025 FUTURE DIRECTIONS

66026 None.

66027 SEE ALSO

66028 *tcsetattr()*

66029 XBD Chapter 11 (on page 199), [<termios.h>](#)

66030 CHANGE HISTORY

66031 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

66032 Issue 6

66033 In the DESCRIPTION, the rate returned as the input baud rate shall be the output rate.
66034 Previously, the number zero was also allowed but was obsolescent.

NAME

tcgetpgrp — get the foreground process group ID

SYNOPSIS

```
#include <unistd.h>

pid_t tcgetpgrp(int fildes);
```

DESCRIPTION

The *tcgetpgrp()* function shall return the value of the process group ID of the foreground process group associated with the terminal.

If there is no foreground process group, *tcgetpgrp()* shall return a value greater than 1 that does not match the process group ID of any existing process group.

The *tcgetpgrp()* function is allowed from a process that is a member of a background process group; however, the information may be subsequently changed by a process that is a member of a foreground process group.

RETURN VALUE

Upon successful completion, *tcgetpgrp()* shall return the value of the process group ID of the foreground process associated with the terminal. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcgetpgrp()* function shall fail if:

[EBADF]	The <i>fildes</i> argument is not a valid file descriptor.
[ENOTTY]	The calling process does not have a controlling terminal, or the file is not the controlling terminal.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setsid(), *setpgid()*, *tcsetpgrp()*
XBD [<sys/types.h>](#), [<unistd.h>](#)

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

tcgetpgrp()*System Interfaces*66077
66078

- In the DESCRIPTION, text previously conditional on support for _POSIX_JOB_CONTROL is now mandatory. This is a FIPS requirement.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

tcgetsid()**NAME**

tcgetsid — get the process group ID for the session leader for the controlling terminal

SYNOPSIS

```
#include <termios.h>

pid_t tcgetsid(int fildes);
```

DESCRIPTION

The *tcgetsid()* function shall obtain the process group ID of the session for which the terminal specified by *fildes* is the controlling terminal.

RETURN VALUE

Upon successful completion, *tcgetsid()* shall return the process group ID of the session associated with the terminal. Otherwise, a value of (**pid_t**)-1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcgetsid()* function shall fail if:

[EBADF]	The <i>fildes</i> argument is not a valid file descriptor.
[ENOTTY]	The calling process does not have a controlling terminal, or the file is not the controlling terminal.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD [<termios.h>](#)

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The [EACCES] error has been removed from the list of mandatory errors, and the description of [ENOTTY] has been reworded.

Issue 7

SD5-XSH-ERN-180 is applied, clarifying the RETURN VALUE section.

The *tcgetsid()* function is moved from the XSI option to the Base.

tcsendbreak()

System Interfaces

66115 **NAME**66116 **tcsendbreak** — send a break for a specific duration66117 **SYNOPSIS**66118 `#include <termios.h>`66119 `int tcsendbreak(int fildev, int duration);`66120 **DESCRIPTION**

66121 If the terminal is using asynchronous serial data transmission, *tcsendbreak()* shall cause
 66122 transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is 0, it
 66123 shall cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5
 66124 seconds. If *duration* is not 0, it shall send zero-valued bits for an implementation-defined period
 66125 of time.

66126 The *fildev* argument is an open file descriptor associated with a terminal.

66127 If the terminal is not using asynchronous serial data transmission, it is implementation-defined
 66128 whether *tcsendbreak()* sends data to generate a break condition or returns without taking any
 66129 action.

66130 Attempts to use *tcsendbreak()* from a process which is a member of a background process group
 66131 on a *fildev* associated with its controlling terminal shall cause the process group to be sent a
 66132 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process
 66133 shall be allowed to perform the operation, and no signal is sent.

66134 **RETURN VALUE**

66135 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 66136 indicate the error.

66137 **ERRORS**

66138 The *tcsendbreak()* function shall fail if:

66139 [EBADF] The *fildev* argument is not a valid file descriptor.

66140 [ENOTTY] The file associated with *fildev* is not a terminal.

66141 The *tcsendbreak()* function may fail if:

66142 [EIO] The process group of the writing process is orphaned, and the writing process
 66143 is not ignoring or blocking SIGTTOU.

66144 **EXAMPLES**

66145 None.

66146 **APPLICATION USAGE**

66147 None.

66148 **RATIONALE**

66149 None.

66150 **FUTURE DIRECTIONS**

66151 None.

66152 **SEE ALSO**

66153 XBD [Chapter 11](#) (on page 199), `<termios.h>`

66154 **CHANGE HISTORY**

66155 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

66156 **Issue 6**

66157 The following new requirements on POSIX implementations derive from alignment with the
66158 Single UNIX Specification:

- 66159 • In the DESCRIPTION, text previously conditional on _POSIX_JOB_CONTROL is now
66160 mandated. This is a FIPS requirement.
- 66161 • The [EIO] error is added.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

tcsetattr()*System Interfaces***NAME**

tcsetattr — set the parameters associated with the terminal

SYNOPSIS

```
#include <termios.h>

int tcsetattr(int fildev, int optional_actions,
              const struct termios *termios_p);
```

DESCRIPTION

The **tcsetattr()** function shall set the parameters associated with the terminal referred to by the open file descriptor *fildev* (an open file descriptor associated with a terminal) from the **termios** structure referenced by *termios_p* as follows:

- If *optional_actions* is TCSANOW, the change shall occur immediately.
- If *optional_actions* is TCSADRAIN, the change shall occur after all output written to *fildev* is transmitted. This function should be used when changing parameters that affect output.
- If *optional_actions* is TCSAFLUSH, the change shall occur after all output written to *fildev* is transmitted, and all input so far received but not read shall be discarded before the change is made.

If the output baud rate stored in the **termios** structure pointed to by *termios_p* is the zero baud rate, B0, the modem control lines shall no longer be asserted. Normally, this shall disconnect the line.

If the input baud rate stored in the **termios** structure pointed to by *termios_p* is 0, the input baud rate given to the hardware is the same as the output baud rate stored in the **termios** structure.

The **tcsetattr()** function shall return successfully if it was able to perform any of the requested actions, even if some of the requested actions could not be performed. It shall set all the attributes that the implementation supports as requested and leave all the attributes not supported by the implementation unchanged. If no part of the request can be honored, it shall return **-1** and set *errno* to **[EINVAL]**. If the input and output baud rates differ and are a combination that is not supported, neither baud rate shall be changed. A subsequent call to **tcgetattr()** shall return the actual state of the terminal device (reflecting both the changes made and not made in the previous **tcsetattr()** call). The **tcsetattr()** function shall not change the values found in the **termios** structure under any circumstances.

The effect of **tcsetattr()** is undefined if the value of the **termios** structure pointed to by *termios_p* was not derived from the result of a call to **tcgetattr()** on *fildev*; an application should modify only fields and flags defined by this volume of POSIX.1-2008 between the call to **tcgetattr()** and **tcsetattr()**, leaving all other fields and flags unmodified.

No actions defined by this volume of POSIX.1-2008, other than a call to **tcsetattr()**, a close of the last file descriptor in the system associated with this terminal device, or an open of the first file descriptor in the system associated with this terminal device (using the **O_TTY_INIT** flag if it is non-zero and the device is not a pseudo-terminal), shall cause any of the terminal attributes defined by this volume of POSIX.1-2008 to change.

If **tcsetattr()** is called from a process which is a member of a background process group on a *fildev* associated with its controlling terminal:

- If the calling process is blocking or ignoring **SIGTTOU** signals, the operation completes normally and no signal is sent.

- Otherwise, a SIGTTOU signal shall be sent to the process group.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcsetattr()* function shall fail if:

- [EBADF] The *fildev* argument is not a valid file descriptor.
- [EINTR] A signal interrupted *tcsetattr()*.
- [EINVAL] The *optional_actions* argument is not a supported value, or an attempt was made to change an attribute represented in the **termios** structure to an unsupported value.
- [ENOTTY] The file associated with *fildev* is not a terminal.

The *tcsetattr()* function may fail if:

- [EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

EXAMPLES

None.

APPLICATION USAGE

If trying to change baud rates, applications should call *tcsetattr()* then call *tcgetattr()* in order to determine what baud rates were actually selected.

In general, there are two reasons for an application to change the parameters associated with a terminal device:

1. The device already has working parameter settings but the application needs a different behavior, such as non-canonical mode instead of canonical mode. The application sets (or clears) only a few flags or *c_cc[]* values. Typically, the terminal device in this case is either the controlling terminal for the process or a pseudo-terminal.
2. The device is a modem or similar piece of equipment connected by a serial line, and it was not open before the application opened it. In this case, the application needs to initialize all of the parameter settings "from scratch". However, since the **termios** structure may include both standard and non-standard parameters, the application cannot just initialize the whole structure in an arbitrary way (e.g., using *memset()*) as this may cause some of the non-standard parameters to be set incorrectly, resulting in non-conforming behavior of the terminal device. Conversely, the application cannot just set the standard parameters, assuming that the non-standard parameters will already have suitable values, as the device might previously have been used with non-conforming parameter settings (and some implementations retain the settings from one use to the next). The solution is to open the terminal device using the *O_TTY_INIT* flag to initialize the terminal device to have conforming parameter settings, obtain those settings using *tcgetattr()*, and then set all of the standard parameters to the desired settings.

RATIONALE

The *tcsetattr()* function can be interrupted in the following situations:

- It is interrupted while waiting for output to drain.

tcsetattr()*System Interfaces*

- It is called from a process in a background process group and SIGTTOU is caught.

See also the RATIONALE section in *tcgetattr()*.

FUTURE DIRECTIONS

Using an input baud rate of 0 to set the input rate equal to the output rate may not necessarily be supported in a future version of this volume of POSIX.1-2008.

SEE ALSO

cfgetispeed(), *tcgetattr()*

XBD Chapter 11 (on page 199), **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now mandated. This is a FIPS requirement.
- The [EIO] error is added.

In the DESCRIPTION, the text describing use of *tcsetattr()* from a process which is a member of a background process group is clarified.

Issue 7

Austin Group Interpretation 1003.1-2001 #144 is applied.

System Interfaces

tcsetpgrp()**NAME**

tcsetpgrp — set the foreground process group ID

SYNOPSIS

```
#include <unistd.h>
```

```
int tcsetpgrp(int fildes, pid_t pgid_id);
```

DESCRIPTION

If the process has a controlling terminal, *tcsetpgrp()* shall set the foreground process group ID associated with the terminal to *pgid_id*. The application shall ensure that the file associated with *fildes* is the controlling terminal of the calling process and the controlling terminal is currently associated with the session of the calling process. The application shall ensure that the value of *pgid_id* matches a process group ID of a process in the same session as the calling process.

Attempts to use *tcsetpgrp()* from a process which is a member of a background process group on a *fildes* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcsetpgrp()* function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EINVAL] This implementation does not support the value in the *pgid_id* argument.

[ENOTTY] The calling process does not have a controlling terminal, or the file is not the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.

[EPERM] The value of *pgid_id* is a value supported by the implementation, but does not match the process group ID of a process in the same session as the calling process.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

tcgetpgrp()

XBD [<sys/types.h>](#), [<unistd.h>](#)

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

tcsetpgrp()*System Interfaces*66308 **Issue 6**

66309 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

66310 The following new requirements on POSIX implementations derive from alignment with the
66311 Single UNIX Specification:

66312 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
66313 required for conforming implementations of previous POSIX specifications, it was not
66314 required for UNIX applications.

66315 • In the DESCRIPTION and ERRORS sections, text previously conditional on
66316 **_POSIX_JOB_CONTROL** is now mandated. This is a FIPS requirement.

66317 The normative text is updated to avoid use of the term “must” for application requirements.

66318 The Open Group Corrigendum U047/4 is applied.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

tdelete()**NAME**

tdelete, tfind, tsearch, twalk — manage a binary search tree

SYNOPSIS

```

#include <search.h>

void *tdelete(const void *restrict key, void **restrict rootp,
              int (*compar)(const void *, const void *));
void *tfind(const void *key, void *const *rootp,
            int (*compar)(const void *, const void *));
void *tsearch(const void *key, void **rootp,
              int (*compar)(const void *, const void *));
void twalk(const void *root,
           void (*action)(const void *, VISIT, int));

```

DESCRIPTION

The *tdelete()*, *tfind()*, *tsearch()*, and *twalk()* functions manipulate binary search trees. Comparisons are made with a user-supplied routine, the address of which is passed as the *compar* argument. This routine is called with two arguments, which are the pointers to the elements being compared. The application shall ensure that the user-supplied routine returns an integer less than, equal to, or greater than 0, according to whether the first argument is to be considered less than, equal to, or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

The *tsearch()* function shall build and access the tree. The *key* argument is a pointer to an element to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed to by *key*, a pointer to this found node shall be returned. Otherwise, the value pointed to by *key* shall be inserted (that is, a new node is created and the value of *key* is copied to this node), and a pointer to this node returned. Only pointers are copied, so the application shall ensure that the calling routine stores the data. The *rootp* argument points to a variable that points to the root node of the tree. A null pointer value for the variable pointed to by *rootp* denotes an empty tree; in this case, the variable shall be set to point to the node which shall be at the root of the new tree.

Like *tsearch()*, *tfind()* shall search for a node in the tree, returning a pointer to it if found. However, if it is not found, *tfind()* shall return a null pointer. The arguments for *tfind()* are the same as for *tsearch()*.

The *tdelete()* function shall delete a node from a binary search tree. The arguments are the same as for *tsearch()*. The variable pointed to by *rootp* shall be changed if the deleted node was the root of the tree. The *tdelete()* function shall return a pointer to the parent of the deleted node, or an unspecified non-null pointer if the deleted node was the root node, or a null pointer if the node is not found.

If *tsearch()* adds an element to a tree, or *tdelete()* successfully deletes an element from a tree, the concurrent use of that tree in another thread, or use of pointers produced by a previous call to *tfind()* or *tsearch()*, produces undefined results.

The *twalk()* function shall traverse a binary search tree. The *root* argument is a pointer to the root node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) The argument *action* is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument shall be the address of the node being visited. The structure pointed to by this argument is unspecified and shall not be modified by the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to-

tdelete()*System Interfaces*

66366 element to access the element stored in the node. The second argument shall be a value from an
 66367 enumeration data type:

66368 `typedef enum { preorder, postorder, endorder, leaf } VISIT;`

66369 (defined in `<search.h>`), depending on whether this is the first, second, or third time that the
 66370 node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a
 66371 leaf. The third argument shall be the level of the node in the tree, with the root being level 0.

66372 If the calling function alters the pointer to the root, the result is undefined.

66373 If the functions pointed to by *action* or *compar* (for any of these binary search functions) change
 66374 the tree, the results are undefined.

66375 These functions are thread-safe only as long as multiple threads do not access the same tree.

RETURN VALUE

66376 If the node is found, both *tsearch()* and *tfind()* shall return a pointer to it. If not, *tfind()* shall
 66377 return a null pointer, and *tsearch()* shall return a pointer to the inserted item.

66379 A null pointer shall be returned by *tsearch()* if there is not enough space available to create a new
 66380 node.

66381 A null pointer shall be returned by *tdelete()*, *tfind()*, and *tsearch()* if *rootp* is a null pointer on
 66382 entry.

66383 The *tdelete()* function shall return a pointer to the parent of the deleted node, or an unspecified
 66384 non-null pointer if the deleted node was the root node, or a null pointer if the node is not found.

66385 The *twalk()* function shall not return a value.

ERRORS

66386 No errors are defined.

EXAMPLES

66389 The following code reads in strings and stores structures containing a pointer to each string and
 66390 a count of its length. It then walks the tree, printing out the stored strings and their lengths in
 66391 alphabetical order.

```

66392 #include <search.h>
66393 #include <string.h>
66394 #include <stdio.h>

66395 #define STRSZ    10000
66396 #define NODSZ    500

66397 struct node {      /* Pointers to these are stored in the tree. */
66398     char    *string;
66399     int     length;
66400 };

66401 char    string_space[STRSZ]; /* Space to store strings. */
66402 struct node nodes[NODSZ];    /* Nodes to store. */
66403 void    *root = NULL;        /* This points to the root. */

66404 int main(int argc, char *argv[])
66405 {
66406     char    *strptr = string_space;
66407     struct node *nodeptr = nodes;
66408     void    print_node(const void *, VISIT, int);

```

```

66409         int    i = 0, node_compare(const void *, const void *);
66410         while (gets(strptr) != NULL && i++ < NODSZ) {
66411             /* Set node. */
66412             nodeptr->string = strptr;
66413             nodeptr->length = strlen(strptr);
66414             /* Put node into the tree. */
66415             (void) tsearch((void *)nodeptr, (void **)&root,
66416                 node_compare);
66417             /* Adjust pointers, so we do not overwrite tree. */
66418             strptr += nodeptr->length + 1;
66419             nodeptr++;
66420         }
66421         twalk(root, print_node);
66422         return 0;
66423     }
66424     /*
66425     * This routine compares two nodes, based on an
66426     * alphabetical ordering of the string field.
66427     */
66428     int
66429     node_compare(const void *node1, const void *node2)
66430     {
66431         return strcmp(((const struct node *) node1)->string,
66432             ((const struct node *) node2)->string);
66433     }
66434     /*
66435     * This routine prints out a node, the second time
66436     * twalk encounters it or if it is a leaf.
66437     */
66438     void
66439     print_node(const void *ptr, VISIT order, int level)
66440     {
66441         const struct node *p = *(const struct node **) ptr;
66442         if (order == postorder || order == leaf) {
66443             (void) printf("string = %s, length = %d\n",
66444                 p->string, p->length);
66445         }
66446     }

```

APPLICATION USAGE

The *root* argument to *twalk()* is one level of indirection less than the *rootp* arguments to *tdelete()* and *tsearch()*.

There are two nomenclatures used to refer to the order in which tree nodes are visited. The *tsearch()* function uses **preorder**, **postorder**, and **endorder** to refer respectively to visiting a node before any of its children, after its left child and before its right, and after both its children. The alternative nomenclature uses **preorder**, **inorder**, and **postorder** to refer to the same visits, which could result in some confusion over the meaning of **postorder**.

Since the return value of *tdelete()* is an unspecified non-null pointer in the case that the root of the tree has been deleted, applications should only use the return value of *tdelete()* as indication

tdelete()*System Interfaces*

66457 of success or failure and should not assume it can be dereferenced. Some implementations in this
66458 case will return a pointer to the new root of the tree (or to an empty tree if the deleted root node
66459 was the only node in the tree); other implementations return arbitrary non-null pointers.

66460 RATIONALE

66461 None.

66462 FUTURE DIRECTIONS

66463 None.

66464 SEE ALSO

66465 *hcreate()*, *lsearch()*

66466 XBD <*search.h*>

66467 CHANGE HISTORY

66468 First released in Issue 1. Derived from Issue 1 of the SVID.

66469 Issue 5

66470 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
66471 previous issues.

66472 Issue 6

66473 The normative text is updated to avoid use of the term “must” for application requirements.

66474 The **restrict** keyword is added to the *tdelete()* prototype for alignment with the
66475 ISO/IEC 9899:1999 standard.

66476 Issue 7

66477 Austin Group Interpretation 1003.1-2001 #149 is applied, clarifying concurrent use of the tree in
66478 another thread.

66479 Austin Group Interpretation 1003.1-2001 #151 is applied, clarifying behavior for *tdelete()* when
66480 the deleted node is the root node.

66481 Austin Group Interpretation 1003.1-2001 #153 is applied.

System Interfaces

tellmdir()66482 **NAME**

66483 tellmdir — current location of a named directory stream

66484 **SYNOPSIS**

```
66485 XSI #include <dirent.h>
66486 long tellmdir(DIR *dirp);
```

66487 **DESCRIPTION**

66488 The *tellmdir()* function shall obtain the current location associated with the directory stream
 66489 specified by *dirp*.

66490 If the most recent operation on the directory stream was a *seekdir()*, the directory position
 66491 returned from the *tellmdir()* shall be the same as that supplied as a *loc* argument for *seekdir()*.

66492 **RETURN VALUE**

66493 Upon successful completion, *tellmdir()* shall return the current location of the specified directory
 66494 stream.

66495 **ERRORS**

66496 No errors are defined.

66497 **EXAMPLES**

66498 None.

66499 **APPLICATION USAGE**

66500 None.

66501 **RATIONALE**

66502 None.

66503 **FUTURE DIRECTIONS**

66504 None.

66505 **SEE ALSO**

66506 *fdopendir()*, *readdir()*, *seekdir()*

66507 XBD **<dirent.h>**

66508 **CHANGE HISTORY**

66509 First released in Issue 2.

tempnam()

System Interfaces

66510 **NAME**66511 **tempnam** — create a name for a temporary file66512 **SYNOPSIS**

```
66513 OB XSI #include <stdio.h>
66514 char *tempnam(const char *dir, const char *pfx);
```

66515 **DESCRIPTION**66516 The *tempnam()* function shall generate a pathname that may be used for a temporary file.

66517 The *tempnam()* function allows the user to control the choice of a directory. The *dir* argument
 66518 points to the name of the directory in which the file is to be created. If *dir* is a null pointer or
 66519 points to a string which is not a name for an appropriate directory, the path prefix defined as
 66520 P_tmpdir in the <stdio.h> header shall be used. If that directory is not accessible, an
 66521 implementation-defined directory may be used.

66522 Many applications prefer their temporary files to have certain initial letter sequences in their
 66523 names. The *pfx* argument should be used for this. This argument may be a null pointer or point
 66524 to a string of up to five bytes to be used as the beginning of the filename.

66525 Some implementations of *tempnam()* may use *tmpnam()* internally. On such implementations, if
 66526 called more than {TMP_MAX} times in a single process, the behavior is implementation-defined.

66527 **RETURN VALUE**

66528 Upon successful completion, *tempnam()* shall allocate space for a string, put the generated
 66529 pathname in that space, and return a pointer to it. The pointer shall be suitable for use in a
 66530 subsequent call to *free()*. Otherwise, it shall return a null pointer and set *errno* to indicate the
 66531 error.

66532 **ERRORS**66533 The *tempnam()* function shall fail if:

66534 [ENOMEM] Insufficient storage space is available.

66535 **EXAMPLES**66536 **Generating a Pathname**

66537 The following example generates a pathname for a temporary file in directory */tmp*, with the
 66538 prefix *file*. After the filename has been created, the call to *free()* deallocates the space used to
 66539 store the filename.

```
66540 #include <stdio.h>
66541 #include <stdlib.h>
66542 ...
66543 char *directory = "/tmp";
66544 char *fileprefix = "file";
66545 char *file;

66546 file = tempnam(directory, fileprefix);
66547 free(file);
```

66548 **APPLICATION USAGE**

66549 This function only creates pathnames. It is the application's responsibility to create and remove
 66550 the files. Between the time a pathname is created and the file is opened, it is possible for some
 66551 other process to create a file with the same name. Applications may find *tmpfile()* more useful.

66552 Applications should use the *tmpfile()*, *mkdtemp()*, or *mkstemp()* functions instead of the

66553 obsolescent *tempnam()* function.

66554 **RATIONALE**

66555 None.

66556 **FUTURE DIRECTIONS**

66557 The *tempnam()* function may be removed in a future version.

66558 **SEE ALSO**

66559 *fopen()*, *free()*, *open()*, *tmpfile()*, *tmpnam()*, *unlink()*

66560 XBD [<stdio.h>](#)

66561 **CHANGE HISTORY**

66562 First released in Issue 1. Derived from Issue 1 of the SVID.

66563 **Issue 5**

66564 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
66565 previous issues.

66566 **Issue 7**

66567 The *tempnam()* function is marked obsolescent.

tfind()*System Interfaces*66568 **NAME**

66569 tfind — search binary search tree

66570 **SYNOPSIS**

```
66571 XSI      #include <search.h>
66572          void *tfind(const void *key, void *const *rootp,
66573                    int (*compar)(const void *, const void *));
```

66574 **DESCRIPTION**66575 Refer to *tdelete()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

tgamma, tgammaf, tgammal — compute gamma() function

SYNOPSIS

```
#include <math.h>

double tgamma(double x);
float tgammaf(float x);
long double tgammal(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall compute the *gamma*() function of *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return *Gamma(x)*.

If *x* is a negative integer, a domain error may occur and either a NaN (if supported) or an implementation-defined value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur and a NaN shall be returned.

If *x* is ± 0 , *tgamma*(), *tgammaf*(), and *tgammal*() shall return $\pm$ HUGE_VAL, $\pm$ HUGE_VALF, and $\pm$ HUGE_VALL, respectively. On systems that support the IEC 60559 Floating-Point option, a pole error shall occur;

otherwise, a pole error may occur.

If the correct value would cause overflow, a range error shall occur and *tgamma*(), *tgammaf*(), and *tgammal*() shall return $\pm$ HUGE_VAL, $\pm$ HUGE_VALF, or $\pm$ HUGE_VALL, respectively, with the same sign as the correct value of the function.

If *x* is NaN, a NaN shall be returned.

If *x* is +Inf, *x* shall be returned.

If *x* is -Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

ERRORS

These functions shall fail if:

Domain Error The value of *x* is a negative integer, or *x* is -Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error The value of *x* is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

tgamma()*System Interfaces*

66620	Range Error	The value overflows.
66621		If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
66622		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
66623		(<i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the overflow
66624		floating-point exception shall be raised.
66625	These functions may fail if:	
66626	Domain Error	The value of <i>x</i> is a negative integer.
66627		If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
66628		then <i>errno</i> shall be set to [EDOM]. If the integer expression (<i>math_errhandling</i>
66629		& MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
66630		shall be raised.
66631	Pole Error	The value of <i>x</i> is zero.
66632		If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
66633		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
66634		(<i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
66635		floating-point exception shall be raised.
66636	EXAMPLES	
66637	None.	
66638	APPLICATION USAGE	
66639	For IEEE Std 754-1985 double , overflow happens when $0 < x < 1/\text{DBL\_MAX}$, and $171.7 < x$.	
66640	On error, the expressions (<i>math_errhandling</i> & MATH_ERRNO) and (<i>math_errhandling</i> &	
66641	MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.	
66642	RATIONALE	
66643	This function is named <i>tgamma()</i> in order to avoid conflicts with the historical <i>gamma()</i> and	
66644	<i>lgamma()</i> functions.	
66645	FUTURE DIRECTIONS	
66646	It is possible that the error response for a negative integer argument may be changed to a pole	
66647	error and a return value of $\pm\text{Inf}$.	
66648	SEE ALSO	
66649	<i>feclearexcept()</i> , <i>fecetestexcept()</i> , <i>lgamma()</i>	
66650	XBD Section 4.19 (on page 116), <math.h>	
66651	CHANGE HISTORY	
66652	First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.	
66653	IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/65 is applied, correcting the third	
66654	paragraph in the RETURN VALUE section.	
66655	Issue 7	
66656	ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #52 (SD5-XSH-ERN-85) is applied.	

System Interfaces

time()**NAME**

time — get time

SYNOPSIS

#include <time.h>

time_t time(time_t *tloc);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *time()* function shall return the value of time in seconds since the Epoch.

The *tloc* argument points to an area where the return value is also stored. If *tloc* is a null pointer, no value is stored.

RETURN VALUE

Upon successful completion, *time()* shall return the value of time. Otherwise, (**time_t**)-1 shall be returned.

ERRORS

No errors are defined.

EXAMPLES**Getting the Current Time**

The following example uses the *time()* function to calculate the time elapsed, in seconds, since the Epoch, *localtime()* to convert that value to a broken-down time, and *asctime()* to convert the broken-down time values into a printable string.

```
#include <stdio.h>
#include <time.h>

int main(void)
{
    time_t result;

    result = time(NULL);
    printf("%s%ju secs since the Epoch\n",
        asctime(localtime(&result)),
        (uintmax_t)result);
    return(0);
}
```

This example writes the current time to *stdout* in a form like this:

```
Wed Jun 26 10:32:15 1996
835810335 secs since the Epoch
```

time()*System Interfaces***Timing an Event**

The following example gets the current time, prints it out in the user's format, and prints the number of minutes to an event being timed.

```
#include <time.h>
#include <stdio.h>
...
time_t now;
int minutes_to_event;
...
time(&now);
minutes_to_event = ...;
printf("The time is ");
puts(asctime(localtime(&now)));
printf("There are %d minutes to the event.\n",
      minutes_to_event);
...
```

APPLICATION USAGE

None.

RATIONALE

The *time()* function returns a value in seconds (type **time_t**) while *times()* returns a set of values in clock ticks (type **clock_t**). Some historical implementations, such as 4.3 BSD, have mechanisms capable of returning more precise times (see below). A generalized timing scheme to unify these various timing mechanisms has been proposed but not adopted.

Implementations in which **time_t** is a 32-bit signed integer (many historical implementations) fail in the year 2038. POSIX.1-2008 does not address this problem. However, the use of the **time_t** type is mandated in order to ease the eventual fix.

The use of the **<time.h>** header instead of **<sys/types.h>** allows compatibility with the ISO C standard.

Many historical implementations (including Version 7) and the 1984 /usr/group standard use **long** instead of **time_t**. This volume of POSIX.1-2008 uses the latter type in order to agree with the ISO C standard.

4.3 BSD includes *time()* only as an alternate function to the more flexible *gettimeofday()* function.

FUTURE DIRECTIONS

In a future version of this volume of POSIX.1-2008, **time_t** is likely to be required to be capable of representing times far in the future. Whether this will be mandated as a 64-bit type or a requirement that a specific date in the future be representable (for example, 10000 AD) is not yet determined. Systems purchased after the approval of this volume of POSIX.1-2008 should be evaluated to determine whether their lifetime will extend past 2038.

SEE ALSO

asctime(), *clock()*, *ctime()*, *difftime()*, *gettimeofday()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *utime()*

XBD **<time.h>**

CHANGE HISTORY

66735 First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

66737 Extensions beyond the ISO C standard are marked.

66738 The EXAMPLES, RATIONALE, and FUTURE DIRECTIONS sections are added.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

timer_create()

System Interfaces

66740 **NAME**66741 **timer_create** — create a per-process timer66742 **SYNOPSIS**

```

66743 CX      #include <signal.h>
66744          #include <time.h>

66745          int timer_create(clockid_t clockid, struct sigevent *restrict evp,
66746                          timer_t *restrict timerid);

```

66747 **DESCRIPTION**

66748 The *timer_create()* function shall create a per-process timer using the specified clock, *clock_id*, as
 66749 the timing base. The *timer_create()* function shall return, in the location referenced by *timerid*, a
 66750 timer ID of type **timer_t** used to identify the timer in timer requests. This timer ID shall be
 66751 unique within the calling process until the timer is deleted. The particular clock, *clock_id*, is
 66752 defined in **<time.h>**. The timer whose ID is returned shall be in a disarmed state upon return
 66753 from *timer_create()*.

66754 The *evp* argument, if non-NULL, points to a **sigevent** structure. This structure, allocated by the
 66755 application, defines the asynchronous notification to occur as specified in [Section 2.4.1](#) (on page
 66756 484) when the timer expires. If the *evp* argument is NULL, the effect is as if the *evp* argument
 66757 pointed to a **sigevent** structure with the *sigev_notify* member having the value SIGEV_SIGNAL,
 66758 the *sigev_signo* having a default signal number, and the *sigev_value* member having the value of
 66759 the timer ID.

66760 Each implementation shall define a set of clocks that can be used as timing bases for per-process
 66761 timers. All implementations shall support a *clock_id* of CLOCK_REALTIME. If the Monotonic
 66762 Clock option is supported, implementations shall support a *clock_id* of CLOCK_MONOTONIC.

66763 Per-process timers shall not be inherited by a child process across a *fork()* and shall be disarmed
 66764 and deleted by an *exec*.

66765 CPT If _POSIX_CPUTIME is defined, implementations shall support *clock_id* values representing the
 66766 CPU-time clock of the calling process.

66767 TCT If _POSIX_THREAD_CPUTIME is defined, implementations shall support *clock_id* values
 66768 representing the CPU-time clock of the calling thread.

66769 CPT|TCT It is implementation-defined whether a *timer_create()* function will succeed if the value defined
 66770 by *clock_id* corresponds to the CPU-time clock of a process or thread different from the process
 66771 or thread invoking the function.

66772 TSA If *evp->sigev_notify* is SIGEV_THREAD and *sev->sigev_notify_attributes* is not NULL, if the
 66773 attribute pointed to by *sev->sigev_notify_attributes* has a thread stack address specified by a call
 66774 to *pthread_attr_setstack()*, the results are unspecified if the signal is generated more than once.

66775 **RETURN VALUE**

66776 If the call succeeds, *timer_create()* shall return zero and update the location referenced by *timerid*
 66777 to a **timer_t**, which can be passed to the per-process timer calls. If an error occurs, the function
 66778 shall return a value of -1 and set *errno* to indicate the error. The value of *timerid* is undefined if
 66779 an error occurs.

66780 **ERRORS**

66781 The *timer_create()* function shall fail if:

66782 [EAGAIN] The system lacks sufficient signal queuing resources to honor the request.

66783	[EAGAIN]	The calling process has already created all of the timers it is allowed by this implementation.
66784		
66785	[EINVAL]	The specified clock ID is not defined.
66786	CPT TCT [ENOTSUP]	The implementation does not support the creation of a timer attached to the CPU-time clock that is specified by <i>clock_id</i> and associated with a process or thread different from the process or thread invoking <i>timer_create()</i> .
66787		
66788		

EXAMPLES

None.

APPLICATION USAGE

If a timer is created which has *evp->sigev_sigev_notify* set to SIGEV_THREAD and the attribute pointed to by *evp->sigev_notify_attributes* has a thread stack address specified by a call to *pthread_attr_setstack()*, the memory dedicated as a thread stack cannot be recovered. The reason for this is that the threads created in response to a timer expiration are created detached, or in an unspecified way if the thread attribute's *detachstate* is PTHREAD_CREATE_JOINABLE. In neither case is it valid to call *pthread_join()*, which makes it impossible to determine the lifetime of the created thread which thus means the stack memory cannot be reused.

RATIONALE**Periodic Timer Overrun and Resource Allocation**

The specified timer facilities may deliver realtime signals (that is, queued signals) on implementations that support this option. Since realtime applications cannot afford to lose notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it must be possible to ensure that sufficient resources exist to deliver the signal when the event occurs. In general, this is not a difficulty because there is a one-to-one correspondence between a request and a subsequent signal generation. If the request cannot allocate the signal delivery resources, it can fail the call with an [EAGAIN] error.

Periodic timers are a special case. A single request can generate an unspecified number of signals. This is not a problem if the requesting process can service the signals as fast as they are generated, thus making the signal delivery resources available for delivery of subsequent periodic timer expiration signals. But, in general, this cannot be assured—processing of periodic timer signals may “overrun”; that is, subsequent periodic timer expirations may occur before the currently pending signal has been delivered.

Also, for signals, according to the POSIX.1-1990 standard, if subsequent occurrences of a pending signal are generated, it is implementation-defined whether a signal is delivered for each occurrence. This is not adequate for some realtime applications. So a mechanism is required to allow applications to detect how many timer expirations were delayed without requiring an indefinite amount of system resources to store the delayed expirations.

The specified facilities provide for an overrun count. The overrun count is defined as the number of extra timer expirations that occurred between the time a timer expiration signal is generated and the time the signal is delivered. The signal-catching function, if it is concerned with overruns, can retrieve this count on entry. With this method, a periodic timer only needs one “signal queuing resource” that can be allocated at the time of the *timer_create()* function call.

A function is defined to retrieve the overrun count so that an application need not allocate static storage to contain the count, and an implementation need not update this storage asynchronously on timer expirations. But, for some high-frequency periodic applications, the overhead of an additional system call on each timer expiration may be prohibitive. The functions, as defined, permit an implementation to maintain the overrun count in user space,

timer_create()*System Interfaces*

associated with the *timerid*. The *timer_getoverrun()* function can then be implemented as a macro that uses the *timerid* argument (which may just be a pointer to a user space structure containing the counter) to locate the overrun count with no system call overhead. Other implementations, less concerned with this class of applications, can avoid the asynchronous update of user space by maintaining the count in a system structure at the cost of the extra system call to obtain it.

Timer Expiration Signal Parameters

The Realtime Signals Extension option supports an application-specific datum that is delivered to the extended signal handler. This value is explicitly specified by the application, along with the signal number to be delivered, in a **sigevent** structure. The type of the application-defined value can be either an integer constant or a pointer. This explicit specification of the value, as opposed to always sending the timer ID, was selected based on existing practice.

It is common practice for realtime applications (on non-POSIX systems or realtime extended POSIX systems) to use the parameters of event handlers as the case label of a switch statement or as a pointer to an application-defined data structure. Since *timer_ids* are dynamically allocated by the *timer_create()* function, they can be used for neither of these functions without additional application overhead in the signal handler; for example, to search an array of saved timer IDs to associate the ID with a constant or application data structure.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres(), *timer_delete()*, *timer_getoverrun()*

XBD <signal.h>, <time.h>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *timer_create()* function is marked as part of the Timers option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

CPU-time clocks are added for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding the requirement for the CLOCK_MONOTONIC clock under the Monotonic Clock option.

The **restrict** keyword is added to the *timer_create()* prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/138 is applied, updating the DESCRIPTION and APPLICATION USAGE sections to describe the case when a timer is created with the notification method set to SIGEV_THREAD.

Issue 7

The *timer_create()* function is moved from the Timers option to the Base.

NAME

timer_delete — delete a per-process timer

SYNOPSIS

```
#include <time.h>

int timer_delete(timer_t timerid);
```

DESCRIPTION

The *timer_delete()* function deletes the specified timer, *timerid*, previously created by the *timer_create()* function. If the timer is armed when *timer_delete()* is called, the behavior shall be as if the timer is automatically disarmed before removal. The disposition of pending signals for the deleted timer is unspecified.

RETURN VALUE

If successful, the *timer_delete()* function shall return a value of zero. Otherwise, the function shall return a value of -1 and set *errno* to indicate the error.

ERRORS

The *timer_delete()* function may fail if:

[EINVAL] The timer ID specified by *timerid* is not a valid timer ID.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

timer_create()

XBD *<time.h>*

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *timer_delete()* function is marked as part of the Timers option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/139 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

Issue 7

The *timer_delete()* function is moved from the Timers option to the Base.

timer_getoverrun()

System Interfaces

66904 **NAME**

66905 timer_getoverrun, timer_gettime, timer_settime — per-process timers

66906 **SYNOPSIS**

```

66907 CX    #include <time.h>
66908
66908    int timer_getoverrun(timer_t timerid);
66909    int timer_gettime(timer_t timerid, struct itimerspec *value);
66910    int timer_settime(timer_t timerid, int flags,
66911        const struct itimerspec *restrict value,
66912        struct itimerspec *restrict ovalue);

```

66913 **DESCRIPTION**

66914 The *timer_gettime()* function shall store the amount of time until the specified timer, *timerid*,
 66915 expires and the reload value of the timer into the space pointed to by the *value* argument. The
 66916 *it_value* member of this structure shall contain the amount of time before the timer expires, or
 66917 zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if
 66918 the timer was armed with absolute time. The *it_interval* member of *value* shall contain the reload
 66919 value last set by *timer_settime()*.

66920 The *timer_settime()* function shall set the time until the next expiration of the timer specified by
 66921 *timerid* from the *it_value* member of the *value* argument and arm the timer if the *it_value* member
 66922 of *value* is non-zero. If the specified timer was already armed when *timer_settime()* is called, this
 66923 call shall reset the time until next expiration to the *value* specified. If the *it_value* member of *value*
 66924 is zero, the timer shall be disarmed. The effect of disarming or resetting a timer with pending
 66925 expiration notifications is unspecified.

66926 If the flag *TIMER_ABSTIME* is not set in the argument *flags*, *timer_settime()* shall behave as if the
 66927 time until next expiration is set to be equal to the interval specified by the *it_value* member of
 66928 *value*. That is, the timer shall expire in *it_value* nanoseconds from when the call is made. If the
 66929 flag *TIMER_ABSTIME* is set in the argument *flags*, *timer_settime()* shall behave as if the time
 66930 until next expiration is set to be equal to the difference between the absolute time specified by
 66931 the *it_value* member of *value* and the current value of the clock associated with *timerid*. That is,
 66932 the timer shall expire when the clock reaches the value specified by the *it_value* member of *value*.
 66933 If the specified time has already passed, the function shall succeed and the expiration
 66934 notification shall be made.

66935 The reload value of the timer shall be set to the value specified by the *it_interval* member of
 66936 *value*. When a timer is armed with a non-zero *it_interval*, a periodic (or repetitive) timer is
 66937 specified.

66938 Time values that are between two consecutive non-negative integer multiples of the resolution of
 66939 the specified timer shall be rounded up to the larger multiple of the resolution. Quantization
 66940 error shall not cause the timer to expire earlier than the rounded time value.

66941 If the argument *ovalue* is not NULL, the *timer_settime()* function shall store, in the location
 66942 referenced by *ovalue*, a value representing the previous amount of time before the timer would
 66943 have expired, or zero if the timer was disarmed, together with the previous timer reload value.
 66944 Timers shall not expire before their scheduled time.

66945 Only a single signal shall be queued to the process for a given timer at any point in time. When a
 66946 timer for which a signal is still pending expires, no signal shall be queued, and a timer overrun
 66947 shall occur. When a timer expiration signal is delivered to or accepted by a process, the
 66948 *timer_getoverrun()* function shall return the timer expiration overrun count for the specified
 66949 timer. The overrun count returned contains the number of extra timer expirations that occurred
 66950 between the time the signal was generated (queued) and when it was delivered or accepted, up

to but not including an implementation-defined maximum of {DELAYTIMER_MAX}. If the number of such extra expirations is greater than or equal to {DELAYTIMER_MAX}, then the overrun count shall be set to {DELAYTIMER_MAX}. The value returned by *timer_getoverrun()* shall apply to the most recent expiration signal delivery or acceptance for the timer. If no expiration signal has been delivered for the timer, the return value of *timer_getoverrun()* is unspecified.

RETURN VALUE

If the *timer_getoverrun()* function succeeds, it shall return the timer expiration overrun count as explained above.

If the *timer_gettime()* or *timer_settime()* functions succeed, a value of 0 shall be returned.

If an error occurs for any of these functions, the value -1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *timer_settime()* function shall fail if:

[EINVAL] A *value* structure specified a nanosecond value less than zero or greater than or equal to 1 000 million, and the *it_value* member of that structure did not specify zero seconds and nanoseconds.

These functions may fail if:

[EINVAL] The *timerid* argument does not correspond to an ID returned by *timer_create()* but not yet deleted by *timer_delete()*.

The *timer_settime()* function may fail if:

[EINVAL] The *it_interval* member of *value* is not zero and the timer was created with notification by creation of a new thread (*sigev_sigev_notify* was SIGEV_THREAD) and a fixed stack address has been set in the thread attribute pointed to by *sigev_notify_attributes*.

EXAMPLES

None.

APPLICATION USAGE

Using fixed stack addresses is problematic when timer expiration is signaled by the creation of a new thread. Since it cannot be assumed that the thread created for one expiration is finished before the next expiration of the timer, it could happen that two threads use the same memory as a stack at the same time. This is invalid and produces undefined results.

RATIONALE

Practical clocks tick at a finite rate, with rates of 100 hertz and 1 000 hertz being common. The inverse of this tick rate is the clock resolution, also called the clock granularity, which in either case is expressed as a time duration, being 10 milliseconds and 1 millisecond respectively for these common rates. The granularity of practical clocks implies that if one reads a given clock twice in rapid succession, one may get the same time value twice; and that timers must wait for the next clock tick after the theoretical expiration time, to ensure that a timer never returns too soon. Note also that the granularity of the clock may be significantly coarser than the resolution of the data format used to set and get time and interval values. Also note that some implementations may choose to adjust time and/or interval values to exactly match the ticks of the underlying clock.

This volume of POSIX.1-2008 defines functions that allow an application to determine the implementation-supported resolution for the clocks and requires an implementation to

timer_getoverrun()

System Interfaces

66996 document the resolution supported for timers and *nanosleep()* if they differ from the supported
 66997 clock resolution. This is more of a procurement issue than a runtime application issue.

66998 **FUTURE DIRECTIONS**

66999 None.

67000 **SEE ALSO**

67001 *clock_getres()*, *timer_create()*

67002 XBD <time.h>

67003 **CHANGE HISTORY**

67004 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

67005 **Issue 6**

67006 The *timer_getoverrun()*, *timer_gettime()*, and *timer_settime()* functions are marked as part of the
 67007 Timers option.

67008 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 67009 implementation does not support the Timers option.

67010 The [EINVAL] error condition is updated to include the following: “and the *it_value* member of
 67011 that structure did not specify zero seconds and nanoseconds.” This change is for IEEE PASC
 67012 Interpretation 1003.1 #89.

67013 The DESCRIPTION for *timer_getoverrun()* is updated to clarify that “If no expiration signal has
 67014 been delivered for the timer, or if the Realtime Signals Extension is not supported, the return
 67015 value of *timer_getoverrun()* is unspecified”.

67016 The **restrict** keyword is added to the *timer_settime()* prototype for alignment with the
 67017 ISO/IEC 9899:1999 standard.

67018 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/140 is applied, updating the ERRORS
 67019 section so that the mandatory [EINVAL] error (“The *timerid* argument does not correspond to an
 67020 ID returned by *timer_create()* but not yet deleted by *timer_delete()*”) becomes optional.

67021 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/141 is applied, updating the ERRORS
 67022 section to include an optional [EINVAL] error for the case when a timer is created with the
 67023 notification method set to SIGEV_THREAD. APPLICATION USAGE text is also added.

67024 **Issue 7**

67025 The *timer_getoverrun()*, *timer_gettime()*, and *timer_settime()* functions are moved from the Timers
 67026 option to the Base.

67027 Functionality relating to the Realtime Signals Extension option is moved to the Base.

NAME

times — get process and waited-for child process times

SYNOPSIS

```
#include <sys/times.h>

clock_t times(struct tms *buffer);
```

DESCRIPTION

The *times()* function shall fill the **tms** structure pointed to by *buffer* with time-accounting information. The **tms** structure is defined in **<sys/times.h>**.

All times are measured in terms of the number of clock ticks used.

The times of a terminated child process shall be included in the *tms_cutime* and *tms_cstime* elements of the parent when *wait()*, *waitid()*, or *waitpid()* returns the process ID of this terminated child. If a child process has not waited for its children, their times shall not be included in its times.

- The *tms_utime* structure member is the CPU time charged for the execution of user instructions of the calling process.
- The *tms_stime* structure member is the CPU time charged for execution by the system on behalf of the calling process.
- The *tms_cutime* structure member is the sum of the *tms_utime* and *tms_cutime* times of the child processes.
- The *tms_cstime* structure member is the sum of the *tms_stime* and *tms_cstime* times of the child processes.

RETURN VALUE

Upon successful completion, *times()* shall return the elapsed real time, in clock ticks, since an arbitrary point in the past (for example, system start-up time). This point does not change from one invocation of *times()* within the process to another. The return value may overflow the possible range of type **clock_t**. If *times()* fails, (**clock_t**)–1 shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

EXAMPLES**Timing a Database Lookup**

The following example defines two functions, *start_clock()* and *end_clock()*, that are used to time a lookup. It also defines variables of type **clock_t** and **tms** to measure the duration of transactions. The *start_clock()* function saves the beginning times given by the *times()* function. The *end_clock()* function gets the ending times and prints the difference between the two times.

```
#include <sys/times.h>
#include <stdio.h>
...
void start_clock(void);
void end_clock(char *msg);
...
static clock_t st_time;
static clock_t en_time;
static struct tms st_cpu;
```

times()*System Interfaces*

```

67072     static struct tms en_cpu;
67073     ...
67074     void
67075     start_clock()
67076     {
67077         st_time = times(&st_cpu);
67078     }
67079     /* This example assumes that the result of each subtraction
67080        is within the range of values that can be represented in
67081        an integer type. */
67082     void
67083     end_clock(char *msg)
67084     {
67085         en_time = times(&en_cpu);
67086         fputs(msg, stdout);
67087         printf("Real Time: %jd, User Time %jd, System Time %jd\n",
67088             (intmax_t)(en_time - st_time),
67089             (intmax_t)(en_cpu.tms_utime - st_cpu.tms_utime),
67090             (intmax_t)(en_cpu.tms_stime - st_cpu.tms_stime));
67091     }

```

APPLICATION USAGE

Applications should use `sysconf(_SC_CLK_TCK)` to determine the number of clock ticks per second as it may vary from system to system.

RATIONALE

The accuracy of the times reported is intentionally left unspecified to allow implementations flexibility in design, from uniprocessor to multi-processor networks.

The inclusion of times of child processes is recursive, so that a parent process may collect the total times of all of its descendants. But the times of a child are only added to those of its parent when its parent successfully waits on the child. Thus, it is not guaranteed that a parent process can always see the total times of all its descendants; see also the discussion of the term “realtime” in `alarm()`.

If the type `clock_t` is defined to be a signed 32-bit integer, it overflows in somewhat more than a year if there are 60 clock ticks per second, or less than a year if there are 100. There are individual systems that run continuously for longer than that. This volume of POSIX.1-2008 permits an implementation to make the reference point for the returned value be the start-up time of the process, rather than system start-up time.

The term “charge” in this context has nothing to do with billing for services. The operating system accounts for time used in this way. That information must be correct, regardless of how that information is used.

FUTURE DIRECTIONS

None.

SEE ALSO

`alarm()`, `exec`, `fork()`, `sysconf()`, `time()`, `wait()`, `waitid()`

XBD `<sys/times.h>`

CHANGE HISTORY

67116 First released in Issue 1. Derived from Issue 1 of the SVID.
67117

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

timezone()*System Interfaces*67118 **NAME**

67119 timezone — difference from UTC and local standard time

67120 **SYNOPSIS**

```
67121 XSI       #include <time.h>  
67122       extern long timezone;
```

67123 **DESCRIPTION**67124 Refer to *tzset()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

67125 **NAME**

67126 tmpfile — create a temporary file

67127 **SYNOPSIS**

67128 #include <stdio.h>

67129 FILE *tmpfile(void);

67130 **DESCRIPTION**

67131 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 67132 conflict between the requirements described here and the ISO C standard is unintentional. This
 67133 volume of POSIX.1-2008 defers to the ISO C standard.

67134 The *tmpfile()* function shall create a temporary file and open a corresponding stream. The file
 67135 shall be automatically deleted when all references to the file are closed. The file is opened as in
 67136 *fopen()* for update (*w+*), except that implementations may restrict the permissions, either by
 67137 clearing the file mode bits or setting them to the value *S_IRUSR* | *S_IWUSR*.

67138 CX In some implementations, a permanent file may be left behind if the process calling *tmpfile()* is
 67139 killed while it is processing a call to *tmpfile()*.

67140 An error message may be written to standard error if the stream cannot be opened.

67141 **RETURN VALUE**

67142 Upon successful completion, *tmpfile()* shall return a pointer to the stream of the file that is
 67143 CX created. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

67144 **ERRORS**67145 The *tmpfile()* function shall fail if:

67146 CX [EINTR] A signal was caught during *tmpfile()*.

67147 CX [EMFILE] All file descriptors available to the process are currently open.

67148 CX [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

67149 CX [ENFILE] The maximum allowable number of files is currently open in the system.

67150 CX [ENOSPC] The directory or file system which would contain the new file cannot be
 67151 expanded.

67152 CX [EOVERFLOW] The file is a regular file and the size of the file cannot be represented correctly
 67153 in an object of type *off_t*.

67154 The *tmpfile()* function may fail if:

67155 CX [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

67156 CX [ENOMEM] Insufficient storage space is available.

67157 **EXAMPLES**67158 **Creating a Temporary File**

67159 The following example creates a temporary file for update, and returns a pointer to a stream for
 67160 the created file in the *fp* variable.

67161 #include <stdio.h>

67162 ...

67163 FILE *fp;

67164 fp = tmpfile ();

tmpfile()*System Interfaces***67165 APPLICATION USAGE**

67166 It should be possible to open at least {TMP_MAX} temporary files during the lifetime of the
 67167 program (this limit may be shared with *tmpnam()*) and there should be no limit on the number
 67168 simultaneously open other than this limit and any limit on the number of open file descriptors
 67169 or streams ({OPEN_MAX}, {FOPEN_MAX}, {STREAM_MAX}).

67170 RATIONALE

67171 None.

67172 FUTURE DIRECTIONS

67173 None.

67174 SEE ALSO

67175 *fopen()*, *mkdtemp()*, *tmpnam()*, *unlink()*

67176 XBD <stdio.h>

67177 CHANGE HISTORY

67178 First released in Issue 1. Derived from Issue 1 of the SVID.

67179 Issue 5

67180 Large File Summit extensions are added.

67181 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes
 67182 in previous issues.

67183 Issue 6

67184 Extensions beyond the ISO C standard are marked.

67185 The following new requirements on POSIX implementations derive from alignment with the
 67186 Single UNIX Specification:

- 67187 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support
 67188 large files.
- 67189 • The [EMFILE] optional error condition is added.

67190 The APPLICATION USAGE section is added for alignment with the ISO/IEC 9899:1999
 67191 standard.

67192 Issue 7

67193 Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying that implementations may
 67194 restrict the permissions of the file created.

67195 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

67196 SD5-XSH-ERN-149 is applied, adding the mandatory [EMFILE] error condition for
 67197 {STREAM_MAX} streams open.

NAME

tmpnam — create a name for a temporary file

SYNOPSIS

```
#include <stdio.h>

char *tmpnam(char *s);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *tmpnam()* function shall generate a string that is a valid filename and that is not the same as the name of an existing file. The function is potentially capable of generating {TMP_MAX} different strings, but any or all of them may already be in use by existing files and thus not be suitable return values.

The *tmpnam()* function generates a different string each time it is called from the same process, up to {TMP_MAX} times. If it is called more than {TMP_MAX} times, the behavior is implementation-defined.

The implementation shall behave as if no function defined in this volume of POSIX.1-2008, except *tmpnam()*, calls *tmpnam()*.

The *tmpnam()* function need not be thread-safe if called with a NULL parameter.

RETURN VALUE

Upon successful completion, *tmpnam()* shall return a pointer to a string. If no suitable string can be generated, the *tmpnam()* function shall return a null pointer.

If the argument *s* is a null pointer, *tmpnam()* shall leave its result in an internal static object and return a pointer to that object. Subsequent calls to *tmpnam()* may modify the same object. If the argument *s* is not a null pointer, it is presumed to point to an array of at least L_tmpnam **chars**; *tmpnam()* shall write its result in that array and shall return the argument as its value.

ERRORS

No errors are defined.

EXAMPLES**Generating a Filename**

The following example generates a unique filename and stores it in the array pointed to by *ptr*.

```
#include <stdio.h>
...
char filename[L_tmpnam+1];
char *ptr;

ptr = tmpnam(filename);
```

APPLICATION USAGE

This function only creates filenames. It is the application's responsibility to create and remove the files.

Between the time a pathname is created and the file is opened, it is possible for some other process to create a file with the same name. Applications may find *tmpfile()* more useful.

Applications should use the *tmpfile()*, *mkstemp()*, or *mkdtemp()* functions instead of the

tmpnam()*System Interfaces*

67240 obsolescent *tmpnam()* function.

67241 **RATIONALE**

67242 None.

67243 **FUTURE DIRECTIONS**

67244 The *tmpnam()* function may be removed in a future version.

67245 **SEE ALSO**

67246 *fopen()*, *open()*, *mkdtemp()*, *tmpnam()*, *tmpfile()*, *unlink()*

67247 XBD **<stdio.h>**

67248 **CHANGE HISTORY**

67249 First released in Issue 1. Derived from Issue 1 of the SVID.

67250 **Issue 5**

67251 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

67252 **Issue 6**

67253 Extensions beyond the ISO C standard are marked.

67254 The normative text is updated to avoid use of the term “must” for application requirements.

67255 The DESCRIPTION is expanded for alignment with the ISO/IEC 9899:1999 standard.

67256 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/142 is applied, updating the
67257 DESCRIPTION to allow implementations of the *tmpnam()* function to call *tmpnam()*.

67258 **Issue 7**

67259 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *tmpnam()* function
67260 need not be thread-safe if called with a NULL parameter.

67261 The *tmpnam()* function is marked obsolescent.

67262 **NAME**

67263 toascii — translate an integer to a 7-bit ASCII character

67264 **SYNOPSIS**

```
67265 OB XSI #include <ctype.h>
67266 int toascii(int c);
```

67267 **DESCRIPTION**67268 The *toascii()* function shall convert its argument into a 7-bit ASCII character.67269 **RETURN VALUE**67270 The *toascii()* function shall return the value (*c* & 0x7f).67271 **ERRORS**

67272 No errors are returned.

67273 **EXAMPLES**

67274 None.

67275 **APPLICATION USAGE**67276 The *toascii()* function cannot be used portably in a localized application.67277 **RATIONALE**

67278 None.

67279 **FUTURE DIRECTIONS**67280 The *toascii()* function may be removed in a future version.67281 **SEE ALSO**67282 *isascii()*

67283 XBD <ctype.h>

67284 **CHANGE HISTORY**

67285 First released in Issue 1. Derived from Issue 1 of the SVID.

67286 **Issue 7**67287 The *toascii()* function is marked obsolescent.

tolower()

System Interfaces

67288 **NAME**67289 **tolower, tolower_l** — transliterate uppercase characters to lowercase67290 **SYNOPSIS**

```
67291       #include <ctype.h>
67292       int tolower(int c);
67293 CX     int tolower_l(int c, locale_t locale);
```

67294 **DESCRIPTION**

67295 CX For *tolower()*: The functionality described on this reference page is aligned with the ISO C
 67296 standard. Any conflict between the requirements described here and the ISO C standard is
 67297 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

67298 CX The *tolower()* and *tolower_l()* functions have as a domain a type **int**, the value of which is
 67299 representable as an **unsigned char** or the value of EOF. If the argument has any other value, the
 67300 CX behavior is undefined. If the argument of *tolower()* or *tolower_l()* represents an uppercase letter,
 67301 and there exists a corresponding lowercase letter as defined by character type information in the
 67302 CX program locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*), the
 67303 result shall be the corresponding lowercase letter. All other arguments in the domain are
 67304 returned unchanged.

67305 **RETURN VALUE**

67306 CX Upon successful completion, the *tolower()* and *tolower_l()* functions shall return the lowercase
 67307 letter corresponding to the argument passed; otherwise, they shall return the argument
 67308 unchanged.

67309 **ERRORS**

67310 The *tolower_l()* function may fail if:

67311 CX [EINVAL] *locale* is not a valid locale object handle.

67312 **EXAMPLES**

67313 None.

67314 **APPLICATION USAGE**

67315 None.

67316 **RATIONALE**

67317 None.

67318 **FUTURE DIRECTIONS**

67319 None.

67320 **SEE ALSO**

67321 [setlocale\(\)](#), [uselocale\(\)](#)
 67322 XBD Chapter 7 (on page 135), [<ctype.h>](#), [<locale.h>](#)

67323 **CHANGE HISTORY**

67324 First released in Issue 1. Derived from Issue 1 of the SVID.

67325 **Issue 6**

67326 Extensions beyond the ISO C standard are marked.

67327 **Issue 7**

67328 The *tolower_l()* function is added from The Open Group Technical Standard, 2006, Extended API
 67329 Set Part 4.

NAME

`toupper`, `toupper_l` — transliterate lowercase characters to uppercase

SYNOPSIS

```
#include <ctype.h>

int toupper(int c);
CX int toupper_l(int c, locale_t locale);
```

DESCRIPTION

For `toupper()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The `toupper()` and `toupper_l()` functions have as a domain a type `int`, the value of which is representable as an **unsigned char** or the value of EOF. If the argument has any other value, the behavior is undefined.

If the argument of `toupper()` or `toupper_l()` represents a lowercase letter, and there exists a corresponding uppercase letter as defined by character type information in the program locale or in the locale represented by `locale`, respectively (category `LC_CTYPE`), the result shall be the corresponding uppercase letter.

All other arguments in the domain are returned unchanged.

RETURN VALUE

Upon successful completion, `toupper()` and `toupper_l()` shall return the uppercase letter corresponding to the argument passed; otherwise, they shall return the argument unchanged.

ERRORS

The `toupper_l()` function may fail if:

[EINVAL] `locale` is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`setlocale()`, `uselocale()`

XBD Chapter 7 (on page 135), `<ctype.h>`, `<locale.h>`

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

toupper()*System Interfaces***Issue 7**

67369 SD5-XSH-ERN-181 is applied, clarifying the RETURN VALUE section.
67370

67371 The *toupper_l()* function is added from The Open Group Technical Standard, 2006, Extended API
67372 Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

towctrans, towctrans_l — wide-character transliteration

SYNOPSIS

```
#include <wctype.h>

wint_t towctrans(wint_t wc, wctrans_t desc);
wint_t towctrans_l(wint_t wc, wctrans_t desc,
    locale_t locale);
```

DESCRIPTION

For *towctrans()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *towctrans()* and *towctrans_l()* functions shall transliterate the wide-character code *wc* using the mapping described by *desc*.

The current setting of the *LC_CTYPE* category in the current locale of the process or in the locale represented by *locale*, respectively, should be the same as during the call to *wctrans()* or *wctrans_l()* that returned the value *desc*.

If the value of *desc* is invalid (that is, not obtained by a call to *wctrans()* or *desc* is invalidated by a subsequent call to *setlocale()* that has affected category *LC_CTYPE*), the result is unspecified.

If the value of *desc* is invalid (that is, not obtained by a call to *wctrans_l()* with the same locale object *locale*) the result is unspecified.

An application wishing to check for error situations should set *errno* to 0 before calling *towctrans()* or *towctrans_l()*.

If *errno* is non-zero on return, an error has occurred.

RETURN VALUE

If successful, the *towctrans()* and *towctrans_l()* functions shall return the mapped value of *wc* using the mapping described by *desc*. Otherwise, they shall return *wc* unchanged.

ERRORS

These functions may fail if:

[EINVAL] *desc* contains an invalid transliteration descriptor.

The *towctrans_l()* function may fail if:

[EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The strings "tolower" and "toupper" are reserved for the standard mapping names. In the table below, the functions in the left column are equivalent to the functions in the right column.

<i>tolower(wc)</i>	<i>towctrans(wc, wctrans("tolower"))</i>
<i>tolower_l(wc, locale)</i>	<i>towctrans_l(wc, wctrans("tolower"), locale)</i>
<i>toupper(wc)</i>	<i>towctrans(wc, wctrans("toupper"))</i>
<i>toupper_l(wc, locale)</i>	<i>towctrans_l(wc, wctrans("toupper"), locale)</i>

towctrans()*System Interfaces*67413 **RATIONALE**

67414 None.

67415 **FUTURE DIRECTIONS**

67416 None.

67417 **SEE ALSO**67418 *tolower()*, *towupper()*, *wctrans()*67419 XBD *<wctype.h>*67420 **CHANGE HISTORY**

67421 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

67422 **Issue 6**

67423 Extensions beyond the ISO C standard are marked.

67424 **Issue 7**67425 The *towctrans_l()* function is added from The Open Group Technical Standard, 2006, Extended
67426 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

`towlower`, `towlower_l` — transliterate uppercase wide-character code to lowercase

SYNOPSIS

```
#include <wctype.h>
```

```
wint_t tolower(wint_t wc);
```

```
CX wint_t tolower_l(wint_t wc, locale_t locale);
```

DESCRIPTION

CX For `towlower()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

CX The `towlower()` and `towlower_l()` functions have as a domain a type `wint_t`, the value of which the application shall ensure is a character representable as a `wchar_t`, and a wide-character code corresponding to a valid character in the current locale or the value of WEOF. If the argument has any other value, the behavior is undefined. If the argument of `towlower()` or `towlower_l()` represents an uppercase wide-character code, and there exists a corresponding lowercase wide-character code as defined by character type information in the locale of the process or in the locale represented by `locale`, respectively (category `LC_CTYPE`), the result shall be the corresponding lowercase wide-character code. All other arguments in the domain are returned unchanged.

RETURN VALUE

CX Upon successful completion, the `towlower()` and `towlower_l()` functions shall return the lowercase letter corresponding to the argument passed; otherwise, they shall return the argument unchanged.

ERRORS

The `towlower_l()` function may fail if:

CX **[EINVAL]** `locale` is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[`setlocale\(\)`](#), [`uselocale\(\)`](#)

XBD [Chapter 7](#) (on page 135), [`<locale.h>`](#), [`<wctype.h>`](#)

CHANGE HISTORY

First released in Issue 4.

tolower()*System Interfaces*67466 **Issue 5**

67467 The following change has been made in this version for alignment with
67468 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 67469 • The SYNOPSIS has been changed to indicate that this function and associated data types
67470 are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

67471 **Issue 6**

67472 The normative text is updated to avoid use of the term “must” for application requirements.

67473 **Issue 7**

67474 The *tolower_l()* function is added from The Open Group Technical Standard, 2006, Extended
67475 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

towupper, towupper_l — transliterate lowercase wide-character code to uppercase

SYNOPSIS

```
#include <wctype.h>
```

```
wint_t towupper(wint_t wc);
```

```
CX wint_t towupper_l(wint_t wc, locale_t locale);
```

DESCRIPTION

CX For *towupper()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

CX The *towupper()* and *towupper_l()* functions have as a domain a type **wint_t**, the value of which the application shall ensure is a character representable as a **wchar_t**, and a wide-character code corresponding to a valid character in the current locale or the value of WEOF. If the argument has any other value, the behavior is undefined. If the argument of *towupper()* or *towupper_l()* represents a lowercase wide-character code, and there exists a corresponding uppercase wide-character code as defined by character type information in the locale of the process or in the locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the corresponding uppercase wide-character code. All other arguments in the domain are returned unchanged.

RETURN VALUE

CX Upon successful completion, the *towupper()* and *towupper_l()* functions shall return the uppercase letter corresponding to the argument passed. Otherwise, they shall return the argument unchanged.

ERRORS

The *towupper_l()* function may fail if:

CX [EINVAL] *locale* is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale(), *uselocale()*

XBD Chapter 7 (on page 135), *<locale.h>*, *<wctype.h>*

CHANGE HISTORY

First released in Issue 4.

towupper()*System Interfaces*67515 **Issue 5**

67516 The following change has been made in this version for alignment with
67517 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 67518 • The SYNOPSIS has been changed to indicate that this function and associated data types
67519 are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

67520 **Issue 6**

67521 The normative text is updated to avoid use of the term “must” for application requirements.

67522 **Issue 7**

67523 The *towupper_l()* function is added from The Open Group Technical Standard, 2006, Extended
67524 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

67525 NAME

67526 trunc, truncf, trunc1 — round to truncated integer value

67527 SYNOPSIS

67528 #include <math.h>

67529 double trunc(double x);

67530 float truncf(float x);

67531 long double trunc1(long double x);

67532 DESCRIPTION

67533 CX The functionality described on this reference page is aligned with the ISO C standard. Any

67534 conflict between the requirements described here and the ISO C standard is unintentional. This

67535 volume of POSIX.1-2008 defers to the ISO C standard.

67536 These functions shall round their argument to the integer value, in floating format, nearest to but

67537 no larger in magnitude than the argument.

67538 RETURN VALUE

67539 Upon successful completion, these functions shall return the truncated integer value.

67540 MX If x is NaN, a NaN shall be returned.

67541 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

67542 ERRORS

67543 No errors are defined.

67544 EXAMPLES

67545 None.

67546 APPLICATION USAGE

67547 None.

67548 RATIONALE

67549 None.

67550 FUTURE DIRECTIONS

67551 None.

67552 SEE ALSO

67553 XBD <math.h>

67554 CHANGE HISTORY

67555 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

truncate()*System Interfaces*67556 **NAME**

67557 truncate — truncate a file to a specified length

67558 **SYNOPSIS**

67559 #include <unistd.h>

67560 int truncate(const char *path, off_t length);

67561 **DESCRIPTION**67562 The *truncate()* function shall cause the regular file named by *path* to have a size which shall be
67563 equal to *length* bytes.67564 If the file previously was larger than *length*, the extra data is discarded. If the file was previously
67565 shorter than *length*, its size is increased, and the extended area appears as if it were zero-filled.

67566 The application shall ensure that the process has write permission for the file.

67567 XSI If the request would cause the file size to exceed the soft file size limit for the process, the
67568 request shall fail and the implementation shall generate the SIGXFSZ signal for the process.67569 The *truncate()* function shall not modify the file offset for any open file descriptions associated
67570 with the file. Upon successful completion, if the file size is changed, *truncate()* shall mark for
67571 update the last data modification and last file status change timestamps of the file, and the
67572 S_ISUID and S_ISGID bits of the file mode may be cleared.67573 **RETURN VALUE**67574 Upon successful completion, *truncate()* shall return 0. Otherwise, -1 shall be returned, and *errno*
67575 set to indicate the error.67576 **ERRORS**67577 The *truncate()* function shall fail if:

67578 [EINTR] A signal was caught during execution.

67579 [EINVAL] The *length* argument was less than 0.

67580 [EFBIG] or [EINVAL]

67581 The *length* argument was greater than the maximum file size.

67582 [EIO] An I/O error occurred while reading from or writing to a file system.

67583 [EACCES] A component of the path prefix denies search permission, or write permission
67584 is denied on the file.

67585 [EISDIR] The named file is a directory.

67586 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
67587 argument.

67588 [ENAMETOOLONG]

67589 The length of a component of a pathname is longer than {NAME_MAX}.

67590 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.67591 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument
67592 contains at least one non-*<slash>* character and ends with one or more trailing
67593 *<slash>* characters and the last pathname component names an existing file
67594 that is neither a directory nor a symbolic link to a directory.

67595 [EROFS] The named file resides on a read-only file system.

67596 The *truncate()* function may fail if:

67597 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
67598 resolution of the *path* argument.

67599 [ENAMETOOLONG]

67600 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
67601 symbolic link produced an intermediate result with a length that exceeds
67602 {PATH_MAX}.

67603 EXAMPLES

67604 None.

67605 APPLICATION USAGE

67606 None.

67607 RATIONALE

67608 None.

67609 FUTURE DIRECTIONS

67610 None.

67611 SEE ALSO

67612 *open()*

67613 XBD <unistd.h>

67614 CHANGE HISTORY

67615 First released in Issue 4, Version 2.

67616 Issue 5

67617 Moved from X/OPEN UNIX extension to BASE.

67618 Large File Summit extensions are added.

67619 Issue 6

67620 This reference page is split out from the *ftruncate()* reference page.

67621 The normative text is updated to avoid use of the term “must” for application requirements.

67622 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
67623 [ELOOP] error condition is added.

67624 Issue 7

67625 Austin Group Interpretation 1003.1-2001 #143 is applied.

67626 The *truncate()* function is moved from the XSI option to the Base.

67627 Changes are made related to support for finegrained timestamps.

67628 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
67629 pathname exists but is not a directory or a symbolic link to a directory.

truncf()*System Interfaces*67630 **NAME**67631 `truncf, truncf` — round to truncated integer value67632 **SYNOPSIS**67633 `#include <math.h>`67634 `float truncf(float x);`67635 `long double truncf(long double x);`67636 **DESCRIPTION**67637 Refer to *trunc()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***tsearch()**67638 **NAME**

67639 tsearch — search a binary search tree

67640 **SYNOPSIS**

```
67641 XSI      #include <search.h>
67642          void *tsearch(const void *key, void **rootp,
67643                        int (*compar)(const void *, const void *));
```

67644 **DESCRIPTION**67645 Refer to *tdelete()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

ttyname()*System Interfaces*67646 **NAME**67647 `ttyname, ttyname_r` — find the pathname of a terminal67648 **SYNOPSIS**67649 `#include <unistd.h>`67650 `char *ttyname(int fildes);`67651 `int ttyname_r(int fildes, char *name, size_t namesize);`67652 **DESCRIPTION**

67653 The `ttyname()` function shall return a pointer to a string containing a null-terminated pathname
 67654 of the terminal associated with file descriptor *fildes*. The return value may point to static data
 67655 whose content is overwritten by each call.

67656 The `ttyname()` function need not be thread-safe.

67657 The `ttyname_r()` function shall store the null-terminated pathname of the terminal associated
 67658 with the file descriptor *fildes* in the character array referenced by *name*. The array is *namesize*
 67659 characters long and should have space for the name and the terminating null character. The
 67660 maximum length of the terminal name shall be {TTY_NAME_MAX}.

67661 **RETURN VALUE**

67662 Upon successful completion, `ttyname()` shall return a pointer to a string. Otherwise, a null
 67663 pointer shall be returned and *errno* set to indicate the error.

67664 If successful, the `ttyname_r()` function shall return zero. Otherwise, an error number shall be
 67665 returned to indicate the error.

67666 **ERRORS**

67667 The `ttyname()` function may fail if:

67668 [EBADF] The *fildes* argument is not a valid file descriptor.

67669 [ENOTTY] The file associated with the *fildes* argument is not a terminal.

67670 The `ttyname_r()` function may fail if:

67671 [EBADF] The *fildes* argument is not a valid file descriptor.

67672 [ENOTTY] The file associated with the *fildes* argument is not a terminal.

67673 [ERANGE] The value of *namesize* is smaller than the length of the string to be returned
 67674 including the terminating null character.

67675 **EXAMPLES**

67676 None.

67677 **APPLICATION USAGE**

67678 None.

67679 **RATIONALE**

67680 The term “terminal” is used instead of the historical term “terminal device” in order to avoid a
 67681 reference to an undefined term.

67682 The thread-safe version places the terminal name in a user-supplied buffer and returns a non-
 67683 zero value if it fails. The non-thread-safe version may return the name in a static data area that
 67684 may be overwritten by each call.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD <unistd.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The *ttyname_r()* function is included for alignment with the POSIX Threads Extension.

A note indicating that the *ttyname()* function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *ttyname_r()* function is marked as part of the Thread-Safe Functions option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The statement that *errno* is set on error is added.
- The [EBADF] and [ENOTTY] optional error conditions are added.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

The *ttyname_r()* function is moved from the Thread-Safe Functions option to the Base.

twalk()*System Interfaces*67705 **NAME**

67706 twalk — traverse a binary search tree

67707 **SYNOPSIS**

```
67708 XSI #include <search.h>
67709 void twalk(const void *root,
67710           void (*action)(const void *, VISIT, int ));
```

67711 **DESCRIPTION**67712 Refer to *tdelete()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

daylight, timezone, tzname, tzset — set timezone conversion information

SYNOPSIS

```
#include <time.h>
```

```
XSI extern int daylight;
      extern long timezone;
CX    extern char *tzname[2];
      void tzset(void);
```

DESCRIPTION

The `tzset()` function shall use the value of the environment variable `TZ` to set time conversion information used by `ctime()`, `localtime()`, `mktime()`, and `strptime()`. If `TZ` is absent from the environment, implementation-defined default timezone information shall be used.

The `tzset()` function shall set the external variable `tzname` as follows:

```
tzname[0] = "std";
tzname[1] = "dst";
```

where `std` and `dst` are as described in XBD Chapter 8 (on page 173).

The `tzset()` function also shall set the external variable `daylight` to 0 if Daylight Savings Time conversions should never be applied for the timezone in use; otherwise, non-zero. The external variable `timezone` shall be set to the difference, in seconds, between Coordinated Universal Time (UTC) and local standard time.

RETURN VALUE

The `tzset()` function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

Example `TZ` variables and their timezone differences are given in the table below:

<i>TZ</i>	<i>timezone</i>
EST5EDT	5*60*60
GMT0	0*60*60
JST-9	-9*60*60
MET-1MEST	-1*60*60
MST7MDT	7*60*60
PST8PDT	8*60*60

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`ctime()`, `localtime()`, `mktime()`, `strptime()`

XBD Chapter 8 (on page 173), `<time.h>`

tzset()*System Interfaces***CHANGE HISTORY**

67755
67756 First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

67757
67758 The example is corrected.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

ulimit — get and set process limits

SYNOPSIS

```
#include <ulimit.h>

long ulimit(int cmd, ...);
```

DESCRIPTION

The *ulimit()* function shall control process limits. The process limits that can be controlled by this function include the maximum size of a single file that can be written (this is equivalent to using *setrlimit()* with *RLIMIT_FSIZE*). The *cmd* values, defined in *<ulimit.h>*, include:

UL_GETFSIZE Return the file size limit (*RLIMIT_FSIZE*) of the process. The limit shall be in units of 512-byte blocks and shall be inherited by child processes. Files of any size can be read. The return value shall be the integer part of the soft file size limit divided by 512. If the result cannot be represented as a **long**, the result is unspecified.

UL_SETFSIZE Set the file size limit for output operations of the process to the value of the second argument, taken as a **long**, multiplied by 512. If the result would overflow an **rlim_t**, the actual value set is unspecified. Any process may decrease its own limit, but only a process with appropriate privileges may increase the limit. The return value shall be the integer part of the new file size limit divided by 512.

The *ulimit()* function shall not change the setting of *errno* if successful.

As all return values are permissible in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *ulimit()*, and, if it returns *-1*, check to see if *errno* is non-zero.

RETURN VALUE

Upon successful completion, *ulimit()* shall return the value of the requested limit. Otherwise, *-1* shall be returned and *errno* set to indicate the error.

ERRORS

The *ulimit()* function shall fail and the limit shall be unchanged if:

[EINVAL] The *cmd* argument is not valid.

[EPERM] A process not having appropriate privileges attempts to increase its file size limit.

EXAMPLES

None.

APPLICATION USAGE

Since the *ulimit()* function uses type **long** rather than **rlim_t**, this function is not sufficient for file sizes on many current systems. Applications should use the *getrlimit()* or *setrlimit()* functions instead of the obsolescent *ulimit()* function.

RATIONALE

None.

ulimit()

System Interfaces

67799 **FUTURE DIRECTIONS**67800 The *ulimit()* function may be removed in a future version.67801 **SEE ALSO**67802 *exec*, *getrlimit()*, *write()*67803 XBD **<ulimit.h>**67804 **CHANGE HISTORY**

67805 First released in Issue 1. Derived from Issue 1 of the SVID.

67806 **Issue 5**67807 In the description of UL_SETFSIZE, the text is corrected to refer to **rlim_t** rather than the
67808 spurious **rlimit_t**.67809 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.67810 **Issue 7**67811 The *ulimit()* function is marked obsolescent.

NAME

umask — set and get the file mode creation mask

SYNOPSIS

```
#include <sys/stat.h>
```

```
mode_t umask(mode_t cmask);
```

DESCRIPTION

The *umask()* function shall set the file mode creation mask of the process to *cmask* and return the previous value of the mask. Only the file permission bits of *cmask* (see <sys/stat.h>) are used; the meaning of the other bits is implementation-defined.

The file mode creation mask of the process is used to turn off permission bits in the *mode* argument supplied during calls to the following functions:

- *open()*, *openat()*, *creat()*, *mkdir()*, *mkdirat()*, *mkfifo()*, and *mkfifoat()*
- *mknod()*, *mknodat()*
- *mq_open()*
- *sem_open()*

Bit positions that are set in *cmask* are cleared in the mode of the created file.

RETURN VALUE

The file permission bits in the value returned by *umask()* shall be the previous value of the file mode creation mask. The state of any other bits in that value is unspecified, except that a subsequent call to *umask()* with the returned value as *cmask* shall leave the state of the mask the same as its state before the first call, including any unspecified use of those bits.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Unsigned argument and return types for *umask()* were proposed. The return type and the argument were both changed to **mode_t**.

Historical implementations have made use of additional bits in *cmask* for their implementation-defined purposes. The addition of the text that the meaning of other bits of the field is implementation-defined permits these implementations to conform to this volume of POSIX.1-2008.

FUTURE DIRECTIONS

None.

SEE ALSO

creat(), *exec*, *mkdir()*, *mkfifo()*, *mknod()*, *mq_open()*, *open()*, *sem_open()*

XBD <sys/stat.h>, <sys/types.h>

umask()

System Interfaces

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/143 is applied, adding the `mknod()`, `mq_open()`, and `sem_open()` functions to the DESCRIPTION and SEE ALSO sections.

System Interfaces

uname()**NAME**

uname — get the name of the current system

SYNOPSIS

```
#include <sys/utsname.h>

int uname(struct utsname *name);
```

DESCRIPTION

The *uname()* function shall store information identifying the current system in the structure pointed to by *name*.

The *uname()* function uses the **utsname** structure defined in **<sys/utsname.h>**.

The *uname()* function shall return a string naming the current system in the character array *sysname*. Similarly, *nodename* shall contain the name of this node within an implementation-defined communications network. The arrays *release* and *version* shall further identify the operating system. The array *machine* shall contain a name that identifies the hardware that the system is running on.

The format of each member is implementation-defined.

RETURN VALUE

Upon successful completion, a non-negative value shall be returned. Otherwise, *-1* shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The inclusion of the *nodename* member in this structure does not imply that it is sufficient information for interfacing to communications networks.

RATIONALE

The values of the structure members are not constrained to have any relation to the version of this volume of POSIX.1-2008 implemented in the operating system. An application should instead depend on *_POSIX_VERSION* and related constants defined in **<unistd.h>**.

This volume of POSIX.1-2008 does not define the sizes of the members of the structure and permits them to be of different sizes, although most implementations define them all to be the same size: eight bytes plus one byte for the string terminator. That size for *nodename* is not enough for use with many networks.

The *uname()* function originated in System III, System V, and related implementations, and it does not exist in Version 7 or 4.3 BSD. The values it returns are set at system compile time in those historical implementations.

4.3 BSD has *gethostname()* and *gethostid()*, which return a symbolic name and a numeric value, respectively. There are related *sethostname()* and *sethostid()* functions that are used to set the values the other two functions return. The former functions are included in this specification, the latter are not.

FUTURE DIRECTIONS

None.

uname()*System Interfaces*67904 **SEE ALSO**67905 XBD [<sys/utsname.h>](#)67906 **CHANGE HISTORY**

67907 First released in Issue 1. Derived from Issue 1 of the SVID.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

ungetc()**NAME**

ungetc — push byte back into input stream

SYNOPSIS

```
#include <stdio.h>
```

```
int ungetc(int c, FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *ungetc()* function shall push the byte specified by *c* (converted to an **unsigned char**) back onto the input stream pointed to by *stream*. The pushed-back bytes shall be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (with the stream pointed to by *stream*) to a file-positioning function (*fseek()*, *fsetpos()*, or *rewind()*) shall discard any pushed-back bytes for the stream. The external storage corresponding to the stream shall be unchanged.

One byte of push-back shall be provided. If *ungetc()* is called too many times on the same stream without an intervening read or file-positioning operation on that stream, the operation may fail.

If the value of *c* equals that of the macro EOF, the operation shall fail and the input stream shall be left unchanged.

A successful call to *ungetc()* shall clear the end-of-file indicator for the stream. The value of the file-position indicator for the stream after reading or discarding all pushed-back bytes shall be the same as it was before the bytes were pushed back. The file-position indicator is decremented by each successful call to *ungetc()*; if its value was 0 before a call, its value is unspecified after the call.

RETURN VALUE

Upon successful completion, *ungetc()* shall return the byte pushed back after conversion. Otherwise, it shall return EOF.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fseek(), *getc()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*

XBD *<stdio.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

ungetwc()*System Interfaces*67950 **NAME**67951 **ungetwc** — push wide-character code back into the input stream67952 **SYNOPSIS**67953 `#include <stdio.h>`67954 `#include <wchar.h>`67955 `wint_t ungetwc(wint_t wc, FILE *stream);`67956 **DESCRIPTION**

67957 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 67958 conflict between the requirements described here and the ISO C standard is unintentional. This
 67959 volume of POSIX.1-2008 defers to the ISO C standard.

67960 The *ungetwc()* function shall push the character corresponding to the wide-character code
 67961 specified by *wc* back onto the input stream pointed to by *stream*. The pushed-back characters
 67962 shall be returned by subsequent reads on that stream in the reverse order of their pushing. A
 67963 successful intervening call (with the stream pointed to by *stream*) to a file-positioning function
 67964 (*fseek()*, *fsetpos()*, or *rewind()*) discards any pushed-back characters for the stream. The external
 67965 storage corresponding to the stream is unchanged.

67966 At least one character of push-back shall be provided. If *ungetwc()* is called too many times on
 67967 the same stream without an intervening read or file-positioning operation on that stream, the
 67968 operation may fail.

67969 If the value of *wc* equals that of the macro WEOF, the operation shall fail and the input stream
 67970 shall be left unchanged.

67971 A successful call to *ungetwc()* shall clear the end-of-file indicator for the stream. The value of the
 67972 file-position indicator for the stream after reading or discarding all pushed-back characters shall
 67973 be the same as it was before the characters were pushed back. The file-position indicator is
 67974 decremented (by one or more) by each successful call to *ungetwc()*; if its value was 0 before a
 67975 call, its value is unspecified after the call.

67976 **RETURN VALUE**

67977 Upon successful completion, *ungetwc()* shall return the wide-character code corresponding to
 67978 the pushed-back character. Otherwise, it shall return WEOF.

67979 **ERRORS**

67980 The *ungetwc()* function may fail if:

67981 CX [EILSEQ] An invalid character sequence is detected, or a wide-character code does not
 67982 correspond to a valid character.

67983 **EXAMPLES**

67984 None.

67985 **APPLICATION USAGE**

67986 None.

67987 **RATIONALE**

67988 None.

67989 **FUTURE DIRECTIONS**

67990 None.

67991 **SEE ALSO**67992 *fseek()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*67993 XBD *<stdio.h>*, *<wchar.h>*67994 **CHANGE HISTORY**

67995 First released in Issue 4. Derived from the MSE working draft.

67996 **Issue 5**67997 The Optional Header (OH) marking is removed from *<stdio.h>*.67998 **Issue 6**

67999 The [EILSEQ] optional error condition is marked CX.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

unlink()*System Interfaces***NAME**

68001 **unlink, unlinkat** — remove a directory entry relative to directory file descriptor

SYNOPSIS

```
68003 #include <unistd.h>
68004 int unlink(const char *path);
68005 int unlinkat(int fd, const char *path, int flag);
```

DESCRIPTION

68007 The *unlink()* function shall remove a link to a file. If *path* names a symbolic link, *unlink()* shall
 68008 remove the symbolic link named by *path* and shall not affect any file or directory named by the
 68009 contents of the symbolic link. Otherwise, *unlink()* shall remove the link named by the pathname
 68010 pointed to by *path* and shall decrement the link count of the file referenced by the link.

68011 When the file's link count becomes 0 and no process has the file open, the space occupied by the
 68012 file shall be freed and the file shall no longer be accessible. If one or more processes have the file
 68013 open when the last link is removed, the link shall be removed before *unlink()* returns, but the
 68014 removal of the file contents shall be postponed until all references to the file are closed.

68015 The *path* argument shall not name a directory unless the process has appropriate privileges and
 68016 the implementation supports using *unlink()* on directories.

68017 Upon successful completion, *unlink()* shall mark for update the last data modification and last
 68018 file status change timestamps of the parent directory. Also, if the file's link count is not 0, the last
 68019 file status change timestamp of the file shall be marked for update.

68020 The *unlinkat()* function shall be equivalent to the *unlink()* or *rmdir()* function except in the case
 68021 where *path* specifies a relative path. In this case the directory entry to be removed is determined
 68022 relative to the directory associated with the file descriptor *fd* instead of the current working
 68023 directory. If the file descriptor was opened without O_SEARCH, the function shall check
 68024 whether directory searches are permitted using the current permissions of the directory
 68025 underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function
 68026 shall not perform the check.

68027 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
 68028 in *<fcntl.h>*:

68029 AT_REMOVEDIR

68030 Remove the directory entry specified by *fd* and *path* as a directory, not a normal file.

68031 If *unlinkat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 68032 directory is used and the behavior shall be identical to a call to *unlink()* or *rmdir()* respectively,
 68033 depending on whether or not the AT_REMOVEDIR bit is set in *flag*.

RETURN VALUE

68035 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 68036 return -1 and set *errno* to indicate the error. If -1 is returned, the named file shall not be changed.

ERRORS

68038 These functions shall fail and shall not unlink the file if:

68039 68040 68041	[EACCES]	Search permission is denied for a component of the path prefix, or write permission is denied on the directory containing the directory entry to be removed.
68042 68043 68044	[EBUSY]	The file named by the <i>path</i> argument cannot be unlinked because it is being used by the system or another process and the implementation considers this an error.

68045	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
68046		
68047	[ENAMETOOLONG]	
68048		The length of a component of a pathname is longer than {NAME_MAX}.
68049	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
68050	[ENOTDIR]	A component of the <i>path</i> prefix is not a directory, or the <i>path</i> argument contains at least one non- <code><slash></code> character and ends with one or more trailing <code><slash></code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
68051		
68052		
68053		
68054	[EPERM]	The file named by <i>path</i> is a directory, and either the calling process does not have appropriate privileges, or the implementation prohibits using <i>unlink()</i> on directories.
68055		
68056		
68057	XSI [EPERM] or [EACCES]	
68058		The <code>S_ISVTX</code> flag is set on the directory containing the file referred to by the <i>path</i> argument and the process does not satisfy the criteria specified in XBD Section 4.2 (on page 107).
68059		
68060		
68061	[EROFS]	The directory entry to be unlinked is part of a read-only file system.
68062		The <i>unlinkat()</i> function shall fail if:
68063	[EACCES]	<i>fd</i> was not opened with <code>O_SEARCH</code> and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
68064		
68065	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching.
68066		
68067	[EEXIST] or [ENOTEMPTY]	
68068		The <i>flag</i> parameter has the <code>AT_REMOVEDIR</code> bit set and the <i>path</i> argument names a directory that is not an empty directory, or there are hard links to the directory other than dot or a single entry in dot-dot.
68069		
68070		
68071	[ENOTDIR]	The <i>flag</i> parameter has the <code>AT_REMOVEDIR</code> bit set and <i>path</i> does not name a directory.
68072		
68073		These functions may fail and not unlink the file if:
68074	XSI [EBUSY]	The file named by <i>path</i> is a named STREAM.
68075	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
68076		
68077	[ENAMETOOLONG]	
68078		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
68079		
68080		
68081	[ETXTBSY]	The entry to be unlinked is the last directory entry to a pure procedure (shared text) file that is being executed.
68082		

unlink()*System Interfaces*

- 68083 The *unlinkat()* function may fail if:
- 68084 [EINVAL] The value of the *flag* argument is not valid.
- 68085 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
- 68086 file descriptor associated with a directory.

68087 **EXAMPLES**68088 **Removing a Link to a File**

68089 The following example shows how to remove a link to a file named **/home/cnd/mod1** by

68090 removing the entry named **/modules/pass1**.

```
68091 #include <unistd.h>
68092
68093 char *path = "/modules/pass1";
68094 int status;
68095 ...
68096 status = unlink(path);
```

68096 **Checking for an Error**

68097 The following example fragment creates a temporary password lock file named **LOCKFILE**,

68098 which is defined as **/etc/ptmp**, and gets a file descriptor for it. If the file cannot be opened for

68099 writing, *unlink()* is used to remove the link between the file descriptor and **LOCKFILE**.

```
68100 #include <sys/types.h>
68101 #include <stdio.h>
68102 #include <fcntl.h>
68103 #include <errno.h>
68104 #include <unistd.h>
68105 #include <sys/stat.h>
68106
68107 #define LOCKFILE "/etc/ptmp"
68108
68109 int pfd; /* Integer for file descriptor returned by open call. */
68110 FILE *fpfd; /* File pointer for use in putpwent(). */
68111 ...
68112 /* Open password Lock file. If it exists, this is an error. */
68113 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL, S_IRUSR
68114 | S_IWUSR | S_IRGRP | S_IROTH)) == -1) {
68115     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
68116     exit(1);
68117 }
68118
68119 /* Lock file created; proceed with fdopen of lock file so that
68120 putpwent() can be used.
68121 */
68122 if ((fpfd = fdopen(pfd, "w")) == NULL) {
68123     close(pfd);
68124     unlink(LOCKFILE);
68125     exit(1);
68126 }
68127 }
```


Replacing Files

The following example fragment uses *unlink()* to discard links to files, so that they can be replaced with new versions of the files. The first call removes the link to **LOCKFILE** if an error occurs. Successive calls remove the links to **SAVEFILE** and **PASSWDFILE** so that new links can be created, then removes the link to **LOCKFILE** when it is no longer needed.

```
#include <sys/types.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>

#define LOCKFILE "/etc/ptmp"
#define PASSWDFILE "/etc/passwd"
#define SAVEFILE "/etc/opasswd"
...
/* If no change was made, assume error and leave passwd unchanged. */
if (!valid_change) {
    fprintf(stderr, "Could not change password for user %s\n", user);
    unlink(LOCKFILE);
    exit(1);
}

/* Change permissions on new password file. */
chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);

/* Remove saved password file. */
unlink(SAVEFILE);

/* Save current password file. */
link(PASSWDFILE, SAVEFILE);

/* Remove current password file. */
unlink(PASSWDFILE);

/* Save new password file as current password file. */
link(LOCKFILE, PASSWDFILE);

/* Remove lock file. */
unlink(LOCKFILE);

exit(0);
```

APPLICATION USAGE

Applications should use *rmdir()* to remove a directory.

RATIONALE

Unlinking a directory is restricted to the superuser in many historical implementations for reasons given in *link()* (see also *rename()*).

The meaning of [EBUSY] in historical implementations is “mount point busy”. Since this volume of POSIX.1-2008 does not cover the system administration concepts of mounting and unmounting, the description of the error was changed to “resource busy”. (This meaning is used by some device drivers when a second process tries to open an exclusive use device.) The wording is also intended to allow implementations to refuse to remove a directory if it is the root or current working directory of any process.

unlink()*System Interfaces*

The standard developers reviewed TR 24715-2006 and noted that LSB-conforming implementations may return [EISDIR] instead of [EPERM] when unlinking a directory. A change to permit this behavior by changing the requirement for [EPERM] to [EPERM] or [EISDIR] was considered, but decided against since it would break existing strictly conforming and conforming applications. Applications written for portability to both POSIX.1-2008 and the LSB should be prepared to handle either error code.

The purpose of the *unlinkat()* function is to remove directory entries in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *unlink()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *unlinkat()* function it can be guaranteed that the removed directory entry is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

close(), *link()*, *remove()*, *rename()*, *rmdir()*, *symlink()*

XBD Section 4.2 (on page 107), *<fcntl.h>*, *<unistd.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The [EBUSY] error is added to the optional part of the ERRORS section.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

The normative text is updated to avoid use of the term “must” for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for operations when the S_ISVTX bit is set.

Text arising from the LSB Conflicts TR is added to the RATIONALE about the use of [EPERM] and [EISDIR].

The *unlinkat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

Changes are made to allow a directory to be opened for searching.

68209
68210

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

unlockpt()

System Interfaces

68211 **NAME**

68212 unlockpt — unlock a pseudo-terminal master/slave pair

68213 **SYNOPSIS**

```
68214 XSI      #include <stdlib.h>
68215          int unlockpt(int fildes);
```

68216 **DESCRIPTION**

68217 The *unlockpt()* function shall unlock the slave pseudo-terminal device associated with the master to which *fildes* refers.

68219 Conforming applications shall ensure that they call *unlockpt()* before opening the slave side of a pseudo-terminal device.

68221 **RETURN VALUE**

68222 Upon successful completion, *unlockpt()* shall return 0. Otherwise, it shall return -1 and set *errno* to indicate the error.

68224 **ERRORS**

68225 The *unlockpt()* function may fail if:

68226 [EBADF] The *fildes* argument is not a file descriptor open for writing.

68227 [EINVAL] The *fildes* argument is not associated with a master pseudo-terminal device.

68228 **EXAMPLES**

68229 None.

68230 **APPLICATION USAGE**

68231 None.

68232 **RATIONALE**

68233 None.

68234 **FUTURE DIRECTIONS**

68235 None.

68236 **SEE ALSO**

68237 *grantpt()*, *open()*, *ptsname()*

68238 XBD <stdlib.h>

68239 **CHANGE HISTORY**

68240 First released in Issue 4, Version 2.

68241 **Issue 5**

68242 Moved from X/OPEN UNIX extension to BASE.

68243 **Issue 6**

68244 The normative text is updated to avoid use of the term “must” for application requirements.

NAME

unsetenv — remove an environment variable

SYNOPSIS

```
#include <stdlib.h>

int unsetenv(const char *name);
```

DESCRIPTION

The *unsetenv()* function shall remove an environment variable from the environment of the calling process. The *name* argument points to a string, which is the name of the variable to be removed. The named argument shall not contain an '=' character. If the named variable does not exist in the current environment, the environment shall be unchanged and the function is considered to have completed successfully.

If the application modifies *environ* or the pointers to which it points, the behavior of *unsetenv()* is undefined. The *unsetenv()* function shall update the list of pointers to which *environ* points.

The *unsetenv()* function need not be thread-safe.

RETURN VALUE

Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to indicate the error, and the environment shall be unchanged.

ERRORS

The *unsetenv()* function shall fail if:

[EINVAL] The *name* argument is a null pointer, points to an empty string, or points to a string containing an '=' character.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to the RATIONALE section in *setenv()*.

FUTURE DIRECTIONS

None.

SEE ALSO

getenv(), *setenv()*

XBD *<stdlib.h>*, *<sys/types.h>*

CHANGE HISTORY

First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Issue 7

Austin Group Interpretation 1003.1-2001 #156 is applied.

uselocale()*System Interfaces*68281 **NAME**

68282 uselocale — use locale in current thread

68283 **SYNOPSIS**

```
68284 CX      #include <locale.h>
68285          locale_t uselocale(locale_t newloc);
```

68286 **DESCRIPTION**

68287 The *uselocale()* function shall set the current locale for the current thread to the locale
 68288 represented by *newloc*.

68289 The value for the *newloc* argument shall be one of the following:

- 68290 1. A value returned by the *newlocale()* or *duplocale()* functions
- 68291 2. The special locale object descriptor *LC_GLOBAL_LOCALE*
- 68292 3. **(locale_t)0**

68293 Once the *uselocale()* function has been called to install a thread-local locale, the behavior of every
 68294 interface using data from the current locale shall be affected for the calling thread. The current
 68295 locale for other threads shall remain unchanged.

68296 If the *newloc* argument is a null pointer, the object returned is the current locale or
 68297 *LC_GLOBAL_LOCALE* if there has been no previous call to *uselocale()* for the current thread.

68298 If the *newloc* argument is *LC_GLOBAL_LOCALE*, the thread shall use the global locale
 68299 determined by the *setlocale()* function.

68300 **RETURN VALUE**

68301 The *uselocale()* function returns the locale handle from the previous call for the current thread. If
 68302 there was no such previous call, the function shall return the value *LC_GLOBAL_LOCALE*.

68303 **ERRORS**

68304 The *uselocale()* function may fail if:

68305 [EINVAL] *locale* is not a valid locale object.

68306 **EXAMPLES**

68307 None.

68308 **APPLICATION USAGE**

68309 Unlike the *setlocale()* function, the *uselocale()* function does not allow replacing some locale
 68310 categories only. Applications that need to install a locale which differs only in a few categories
 68311 must use *newlocale()* to change a locale object equivalent to the currently used locale and install
 68312 it.

68313 **RATIONALE**

68314 None.

68315 **FUTURE DIRECTIONS**

68316 None.

68317 **SEE ALSO**

68318 *duplocale()*, *freelocale()*, *newlocale()*, *setlocale()*

68319 XBD **<locale.h>**

68320	CHANGE HISTORY
68321	First released in Issue 7.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

utime()*System Interfaces*68322 **NAME**68323 **utime** — set file access and modification times68324 **SYNOPSIS**

```
68325 OB #include <utime.h>
68326 int utime(const char *path, const struct utimbuf *times);
```

68327 **DESCRIPTION**

68328 The *utime()* function shall set the access and modification times of the file named by the *path*
 68329 argument.

68330 If *times* is a null pointer, the access and modification times of the file shall be set to the current
 68331 time. The effective user ID of the process shall match the owner of the file, or the process has
 68332 write permission to the file or has appropriate privileges, to use *utime()* in this manner.

68333 If *times* is not a null pointer, *times* shall be interpreted as a pointer to a **utimbuf** structure and the
 68334 access and modification times shall be set to the values contained in the designated structure.
 68335 Only a process with the effective user ID equal to the user ID of the file or a process with
 68336 appropriate privileges may use *utime()* this way.

68337 The **utimbuf** structure is defined in the **<utime.h>** header. The times in the structure **utimbuf**
 68338 are measured in seconds since the Epoch.

68339 Upon successful completion, the *utime()* function shall mark the last file status change
 68340 timestamp for update; see **<sys/stat.h>**.

68341 **RETURN VALUE**

68342 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* shall
 68343 be set to indicate the error, and the file times shall not be affected.

68344 **ERRORS**

68345 The *utime()* function shall fail if:

68346 [EACCES] Search permission is denied by a component of the path prefix; or the *times*
 68347 argument is a null pointer and the effective user ID of the process does not
 68348 match the owner of the file, the process does not have write permission for the
 68349 file, and the process does not have appropriate privileges.

68350 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 68351 argument.

68352 [ENAMETOOLONG]

68353 The length of a component of a pathname is longer than {NAME_MAX}.

68354 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

68355 [ENOTDIR] A component of the path prefix is not a directory, or the *path* argument
 68356 contains at least one non-*<slash>* character and ends with one or more trailing
 68357 *<slash>* characters and the last pathname component names an existing file
 68358 that is neither a directory nor a symbolic link to a directory.

68359 [EPERM] The *times* argument is not a null pointer and the effective user ID of the calling
 68360 process does not match the owner of the file and the calling process does not
 68361 have appropriate privileges.

68362 [EROFS] The file system containing the file is read-only.

68363 The *utime()* function may fail if:

68364 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
68365 resolution of the *path* argument.

68366 [ENAMETOOLONG]

68367 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
68368 symbolic link produced an intermediate result with a length that exceeds
68369 {PATH_MAX}.

68370 EXAMPLES

68371 None.

68372 APPLICATION USAGE

68373 Since the **utimbuf** structure only contains **time_t** variables and is not accurate to fractions of a
68374 second, applications should use the *utimensat()* function instead of the obsolescent *utime()*
68375 function.

68376 RATIONALE

68377 The *actime* structure member must be present so that an application may set it, even though an
68378 implementation may ignore it and not change the last data access timestamp on the file. If an
68379 application intends to leave one of the times of a file unchanged while changing the other, it
68380 should use *stat()* or *fstat()* to retrieve the file's *st_atim* and *st_mtim* parameters, set *actime* and
68381 *modtime* in the buffer, and change one of them before making the *utime()* call.

68382 FUTURE DIRECTIONS

68383 The *utime()* function may be removed in a future version.

68384 SEE ALSO

68385 *fstat()*, *fstatat()*, *futimens()*

68386 XBD <sys/stat.h>, <utime.h>

68387 CHANGE HISTORY

68388 First released in Issue 1. Derived from Issue 1 of the SVID.

68389 Issue 6

68390 The following new requirements on POSIX implementations derive from alignment with the
68391 Single UNIX Specification:

- 68392 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
68393 required for conforming implementations of previous POSIX specifications, it was not
68394 required for UNIX applications.
- 68395 • The [ELOOP] mandatory error condition is added.
- 68396 • A second [ENAMETOOLONG] is added as an optional error condition.

68397 The following changes were made to align with the IEEE P1003.1a draft standard:

- 68398 • The [ELOOP] optional error condition is added.

68399 The normative text is updated to avoid use of the term “must” for application requirements.

68400 Issue 7

68401 Austin Group Interpretation 1003.1-2001 #143 is applied.

68402 The *utime()* function is marked obsolescent.

68403 Changes are made related to support for finegrained timestamps.

utimensat()*System Interfaces*68404 **NAME**

68405 utimensat, utimes — set file access and modification times relative to directory file descriptor

68406 **SYNOPSIS**

68407 #include <sys/stat.h>

68408 int utimensat(int *fd*, const char **path*, const struct timespec *times*[2],68409 int *flag*);

68410 XSI #include <sys/time.h>

68411 int utimes(const char **path*, const struct timeval *times*[2]);68412 **DESCRIPTION**68413 Refer to *futimens()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***va_arg()**68414 **NAME**

68415 va_arg, va_copy, va_end, va_start — handle variable argument list

68416 **SYNOPSIS**

```
68417       #include <stdarg.h>
68418       type va_arg(va_list ap, type);
68419       void va_copy(va_list dest, va_list src);
68420       void va_end(va_list ap);
68421       void va_start(va_list ap, argN);
```

68422 **DESCRIPTION**68423 Refer to XBD [<stdarg.h>](#)

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

fprintf()68424 **NAME**68425 `vdprintf, fprintf, vprintf, vsnprintf, vsprintf` — format output of a stdarg argument list68426 **SYNOPSIS**68427 `#include <stdarg.h>`68428 `#include <stdio.h>`

```

68429 CX int vdprintf(int fildes, const char *restrict format, va_list ap);
68430 int fprintf(FILE *restrict stream, const char *restrict format,
68431 va_list ap);
68432 int vprintf(const char *restrict format, va_list ap);
68433 int vsnprintf(char *restrict s, size_t n, const char *restrict format,
68434 va_list ap);
68435 int vsprintf(char *restrict s, const char *restrict format, va_list ap);

```

68436 **DESCRIPTION**

68437 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 68438 conflict between the requirements described here and the ISO C standard is unintentional. This
 68439 volume of POSIX.1-2008 defers to the ISO C standard.

68440 CX The `vdprintf()`, `fprintf()`, `vprintf()`, `vsnprintf()`, and `vsprintf()` functions shall be equivalent to the
 68441 CX `dprintf()`, `fprintf()`, `printf()`, `snprintf()`, and `sprintf()` functions respectively, except that instead of
 68442 being called with a variable number of arguments, they are called with an argument list as
 68443 defined by `<stdarg.h>`.

68444 These functions shall not invoke the `va_end` macro. As these functions invoke the `va_arg` macro,
 68445 the value of `ap` after the return is unspecified.

68446 **RETURN VALUE**68447 Refer to `fprintf()`.68448 **ERRORS**68449 Refer to `fprintf()`.68450 **EXAMPLES**

68451 None.

68452 **APPLICATION USAGE**68453 Applications using these functions should call `va_end(ap)` afterwards to clean up.68454 **RATIONALE**

68455 None.

68456 **FUTURE DIRECTIONS**

68457 None.

68458 **SEE ALSO**68459 `fprintf()`68460 XBD `<stdarg.h>`, `<stdio.h>`68461 **CHANGE HISTORY**

68462 First released in Issue 1. Derived from Issue 1 of the SVID.

68463 **Issue 5**68464 The `vsnprintf()` function is added.

68465 **Issue 6**

68466 The *vfprintf()*, *vprintf()*, *vsprintf()*, and *vsprintf()* functions are updated for alignment with the
68467 ISO/IEC 9899:1999 standard.

68468 **Issue 7**

68469 The *vdprintf()* function is added to complement the *dprintf()* function from The Open Group
68470 Technical Standard, 2006, Extended API Set Part 1.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

vfscanf()*System Interfaces*68471 **NAME**68472 **vfscanf, vscanf, vsscanf** — format input of a stdarg argument list68473 **SYNOPSIS**68474 `#include <stdarg.h>`68475 `#include <stdio.h>`68476 `int vfscanf(FILE *restrict stream, const char *restrict format,`
68477 `va_list arg);`68478 `int vscanf(const char *restrict format, va_list arg);`68479 `int vsscanf(const char *restrict s, const char *restrict format,`
68480 `va_list arg);`68481 **DESCRIPTION**68482 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
68483 conflict between the requirements described here and the ISO C standard is unintentional. This
68484 volume of POSIX.1-2008 defers to the ISO C standard.68485 The *vscanf()*, *vfscanf()*, and *vsscanf()* functions shall be equivalent to the *scanf()*, *fscanf()*, and
68486 *sscanf()* functions, respectively, except that instead of being called with a variable number of
68487 arguments, they are called with an argument list as defined in the **<stdarg.h>** header. These
68488 functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro, the
68489 value of *ap* after the return is unspecified.68490 **RETURN VALUE**68491 Refer to *fscanf()*.68492 **ERRORS**68493 Refer to *fscanf()*.68494 **EXAMPLES**

68495 None.

68496 **APPLICATION USAGE**68497 Applications using these functions should call *va_end(ap)* afterwards to clean up.68498 **RATIONALE**

68499 None.

68500 **FUTURE DIRECTIONS**

68501 None.

68502 **SEE ALSO**68503 *fscanf()*68504 XBD **<stdarg.h>**, **<stdio.h>**68505 **CHANGE HISTORY**

68506 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

NAME

vfwprintf, vswprintf, vwprintf — wide-character formatted output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vfwprintf(FILE *restrict stream, const wchar_t *restrict format,
va_list arg);
int vswprintf(wchar_t *restrict ws, size_t n,
const wchar_t *restrict format, va_list arg);
int vwprintf(const wchar_t *restrict format, va_list arg);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *vfwprintf()*, *vswprintf()*, and *vwprintf()* functions shall be equivalent to *fprintf()*, *swprintf()*, and *wprintf()* respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by **<stdarg.h>**.

These functions shall not invoke the *va_end* macro. However, as these functions do invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fprintf()*.

ERRORS

Refer to *fprintf()*.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end(ap)* afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf()

XBD **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The *vfwprintf()*, *vswprintf()*, and *vwprintf()* prototypes are updated for alignment with the ISO/IEC 9899:1999 standard. ()

vfwscanf()*System Interfaces*68548 **NAME**68549 **vfwscanf, vswscanf, vwscanf** — wide-character formatted input of a stdarg argument list68550 **SYNOPSIS**68551 `#include <stdarg.h>`68552 `#include <stdio.h>`68553 `#include <wchar.h>`68554 `int vfwscanf(FILE *restrict stream, const wchar_t *restrict format,`
68555 `va_list arg);`68556 `int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,`
68557 `va_list arg);`68558 `int vwscanf(const wchar_t *restrict format, va_list arg);`68559 **DESCRIPTION**68560 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
68561 conflict between the requirements described here and the ISO C standard is unintentional. This
68562 volume of POSIX.1-2008 defers to the ISO C standard.68563 The *vfwscanf()*, *vswscanf()*, and *vwscanf()* functions shall be equivalent to the *fwscanf()*,
68564 *swscanf()*, and *wscanf()* functions, respectively, except that instead of being called with a variable
68565 number of arguments, they are called with an argument list as defined in the **<stdarg.h>** header.
68566 These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro,
68567 the value of *ap* after the return is unspecified.68568 **RETURN VALUE**68569 Refer to *fwscanf()*.68570 **ERRORS**68571 Refer to *fwscanf()*.68572 **EXAMPLES**

68573 None.

68574 **APPLICATION USAGE**68575 Applications using these functions should call *va_end(ap)* afterwards to clean up.68576 **RATIONALE**

68577 None.

68578 **FUTURE DIRECTIONS**

68579 None.

68580 **SEE ALSO**68581 *fwscanf()*68582 XBD **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**68583 **CHANGE HISTORY**

68584 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

*System Interfaces***vprintf()**68585 **NAME**

68586 vprintf — format the output of a stdarg argument list

68587 **SYNOPSIS**

68588 #include <stdarg.h>

68589 #include <stdio.h>

68590 int vprintf(const char *restrict *format*, va_list *ap*);68591 **DESCRIPTION**68592 Refer to *vfprintf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

vscanf()*System Interfaces*68593 **NAME**68594 **vscanf** — format input of a stdarg argument list68595 **SYNOPSIS**68596 `#include <stdarg.h>`68597 `#include <stdio.h>`68598 `int vscanf(const char *restrict format, va_list arg);`68599 **DESCRIPTION**68600 Refer to *vscanf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***vsnprintf()**68601 **NAME**

68602 vsprintf, vsprintf — format output of a stdarg argument list

68603 **SYNOPSIS**

68604 #include <stdarg.h>

68605 #include <stdio.h>

68606 int vsnprintf(char *restrict *s*, size_t *n*,68607 const char *restrict *format*, va_list *ap*);68608 int vsprintf(char *restrict *s*, const char *restrict *format*,68609 va_list *ap*);68610 **DESCRIPTION**68611 Refer to *fprintf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

vsscanf()*System Interfaces*68612 **NAME**68613 `vsscanf` — format input of a stdarg argument list68614 **SYNOPSIS**68615 `#include <stdarg.h>`68616 `#include <stdio.h>`68617 `int vsscanf(const char *restrict s, const char *restrict format,`
68618 `va_list arg);`68619 **DESCRIPTION**68620 Refer to *vfscanf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

68621 **NAME**

68622 vswprintf — wide-character formatted output of a stdarg argument list

68623 **SYNOPSIS**

68624 #include <stdarg.h>

68625 #include <stdio.h>

68626 #include <wchar.h>

```
68627 int vswprintf(wchar_t *restrict ws, size_t n,  
68628             const wchar_t *restrict format, va_list arg);
```

68629 **DESCRIPTION**68630 Refer to *vfprintf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

vswscanf()*System Interfaces*68631 **NAME**68632 **vswscanf** — wide-character formatted input of a stdarg argument list68633 **SYNOPSIS**68634 `#include <stdarg.h>`68635 `#include <stdio.h>`68636 `#include <wchar.h>`68637 `int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,`
68638 `va_list arg);`68639 **DESCRIPTION**68640 Refer to *vfwscanf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***vwprintf()**68641 **NAME**

68642 vwprintf — wide-character formatted output of a stdarg argument list

68643 **SYNOPSIS**

68644 #include <stdarg.h>

68645 #include <stdio.h>

68646 #include <wchar.h>

68647 int vwprintf(const wchar_t *restrict *format*, va_list *arg*);68648 **DESCRIPTION**68649 Refer to *vfwprintf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

vwscanf()*System Interfaces*68650 **NAME**

68651 vwscanf — wide-character formatted input of a stdarg argument list

68652 **SYNOPSIS**

68653 #include <stdarg.h>

68654 #include <stdio.h>

68655 #include <wchar.h>

68656 int vwscanf(const wchar_t *restrict *format*, va_list *arg*);68657 **DESCRIPTION**68658 Refer to *vfwscanf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

wait()**NAME**

wait, waitpid — wait for a child process to stop or terminate

SYNOPSIS

```
#include <sys/wait.h>
```

```
pid_t wait(int *stat_loc);
```

```
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

DESCRIPTION

The *wait()* and *waitpid()* functions shall obtain status information pertaining to one of the caller's child processes. Various options permit status information to be obtained for child processes that have terminated or stopped. If status information is available for two or more child processes, the order in which their status is reported is unspecified.

The *wait()* function shall suspend execution of the calling thread until status information for one of the terminated child processes of the calling process is available, or until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process. If more than one thread is suspended in *wait()* or *waitpid()* awaiting termination of the same process, exactly one thread shall return the process status at the time of the target process termination. If status information is available prior to the call to *wait()*, return shall be immediate.

The *waitpid()* function shall be equivalent to *wait()* if the *pid* argument is **(pid_t)−1** and the *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and *options* arguments.

The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid()* function shall only return the status of a child process from this set:

- If *pid* is equal to **(pid_t)−1**, *status* is requested for any child process. In this respect, *waitpid()* is then equivalent to *wait()*.
- If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is requested.
- If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that of the calling process.
- If *pid* is less than **(pid_t)−1**, *status* is requested for any child process whose process group ID is equal to the absolute value of *pid*.

The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the following flags, defined in the **<sys/wait.h>** header:

WCONTINUED The *waitpid()* function shall report the status of any continued child process specified by *pid* whose status has not been reported since it continued from a job control stop.

WNOHANG The *waitpid()* function shall not suspend execution of the calling thread if *status* is not immediately available for one of the child processes specified by *pid*.

WUNTRACED The status of any child processes specified by *pid* that are stopped, and whose status has not yet been reported since they stopped, shall also be reported to the requesting process.

If the calling process has **SA_NOCLDWAIT** set or has **SIGCHLD** set to **SIG_IGN**, and the process has no unwaited-for children that were transformed into zombie processes, the calling thread shall block until all of the children of the process containing the calling thread terminate, and *wait()* and *waitpid()* shall fail and set *errno* to **[ECHILD]**.

wait()*System Interfaces*

If *wait()* or *waitpid()* return because the status of a child process is available, these functions shall return a value equal to the process ID of the child process. In this case, if the value of the argument *stat_loc* is not a null pointer, information shall be stored in the location pointed to by *stat_loc*. The value stored at the location pointed to by *stat_loc* shall be 0 if and only if the status returned is from a terminated child process that terminated by one of the following means:

1. The process returned 0 from *main()*.
2. The process called *_exit()* or *exit()* with a *status* argument of 0.
3. The process was terminated because the last thread in the process terminated.

Regardless of its value, this information may be interpreted using the following macros, which are defined in **<sys/wait.h>** and evaluate to integral expressions; the *stat_val* argument is the integer value pointed to by *stat_loc*.

WIFEXITED(*stat_val*)

Evaluates to a non-zero value if *status* was returned for a child process that terminated normally.

WEXITSTATUS(*stat_val*)

If the value of WIFEXITED(*stat_val*) is non-zero, this macro evaluates to the low-order 8 bits of the *status* argument that the child process passed to *_exit()* or *exit()*, or the value the child process returned from *main()*.

WIFSIGNALED(*stat_val*)

Evaluates to a non-zero value if *status* was returned for a child process that terminated due to the receipt of a signal that was not caught (see **<signal.h>**).

WTERMSIG(*stat_val*)

If the value of WIFSIGNALED(*stat_val*) is non-zero, this macro evaluates to the number of the signal that caused the termination of the child process.

WIFSTOPPED(*stat_val*)

Evaluates to a non-zero value if *status* was returned for a child process that is currently stopped.

WSTOPSIG(*stat_val*)

If the value of WIFSTOPPED(*stat_val*) is non-zero, this macro evaluates to the number of the signal that caused the child process to stop.

WIFCONTINUED(*stat_val*)

Evaluates to a non-zero value if *status* was returned for a child process that has continued from a job control stop.

It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes created by *posix_spawn()* or *posix_spawnnp()* can indicate a WIFSTOPPED(*stat_val*) before subsequent calls to *wait()* or *waitpid()* indicate WIFEXITED(*stat_val*) as the result of an error detected before the new process image starts executing.

It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes created by *posix_spawn()* or *posix_spawnnp()* can indicate a WIFSIGNALED(*stat_val*) if a signal is sent to the parent's process group after *posix_spawn()* or *posix_spawnnp()* is called.

If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that specified the WUNTRACED flag and did not specify the WCONTINUED flag, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), and WIFSTOPPED(**stat_loc*) shall evaluate to a non-zero value.

System Interfaces

wait()

68748 If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that specified the
 68749 XSI WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(**stat_loc*),
 68750 XSI WIFSIGNALED(**stat_loc*), WIFSTOPPED(**stat_loc*), and WIFCONTINUED(**stat_loc*) shall
 68751 evaluate to a non-zero value.

68752 If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that did not specify the
 68753 XSI WUNTRACED or WCONTINUED flags, or by a call to the *wait()* function, exactly one of the
 68754 macros WIFEXITED(**stat_loc*) and WIFSIGNALED(**stat_loc*) shall evaluate to a non-zero value.

68755 If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that did not specify the
 68756 XSI WUNTRACED flag and specified the WCONTINUED flag, or by a call to the *wait()* function,
 68757 XSI exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), and
 68758 WIFCONTINUED(**stat_loc*) shall evaluate to a non-zero value.

68759 If _POSIX_REALTIME_SIGNALS is defined, and the implementation queues the SIGCHLD
 68760 signal, then if *wait()* or *waitpid()* returns because the status of a child process is available, any
 68761 pending SIGCHLD signal associated with the process ID of the child process shall be discarded.
 68762 Any other pending SIGCHLD signals shall remain pending.

68763 Otherwise, if SIGCHLD is blocked, if *wait()* or *waitpid()* return because the status of a child
 68764 process is available, any pending SIGCHLD signal shall be cleared unless the status of another
 68765 child process is available.

68766 For all other conditions, it is unspecified whether child *status* will be available when a SIGCHLD
 68767 signal is delivered.

68768 There may be additional implementation-defined circumstances under which *wait()* or *waitpid()*
 68769 report *status*. This shall not occur unless the calling process or one of its child processes
 68770 explicitly makes use of a non-standard extension. In these cases the interpretation of the
 68771 reported *status* is implementation-defined.

68772 If a parent process terminates without waiting for all of its child processes to terminate, the
 68773 remaining child processes shall be assigned a new parent process ID corresponding to an
 68774 implementation-defined system process.

68775 **RETURN VALUE**

68776 If *wait()* or *waitpid()* returns because the status of a child process is available, these functions
 68777 shall return a value equal to the process ID of the child process for which *status* is reported. If
 68778 *wait()* or *waitpid()* returns due to the delivery of a signal to the calling process, -1 shall be
 68779 returned and *errno* set to [EINTR]. If *waitpid()* was invoked with WNOHANG set in *options*, it
 68780 has at least one child process specified by *pid* for which *status* is not available, and *status* is not
 68781 available for any process specified by *pid*, 0 is returned. Otherwise, (*pid_t*)-1 shall be returned,
 68782 and *errno* set to indicate the error.

68783 **ERRORS**

68784 The *wait()* function shall fail if:

68785 [ECHILD] The calling process has no existing unwaited-for child processes.

68786 [EINTR] The function was interrupted by a signal. The value of the location pointed to
 68787 by *stat_loc* is undefined.

68788 The *waitpid()* function shall fail if:

68789 [ECHILD] The process specified by *pid* does not exist or is not a child of the calling
 68790 process, or the process group specified by *pid* does not exist or does not have
 68791 any member process that is a child of the calling process.

wait()*System Interfaces*

- 68792 [EINTR] The function was interrupted by a signal. The value of the location pointed to
 68793 by *stat_loc* is undefined.
- 68794 [EINVAL] The *options* argument is not valid.

68795 **EXAMPLES**68796 **Waiting for a Child Process and then Checking its Status**

68797 The following example demonstrates the use of *waitpid()*, *fork()*, and the macros used to
 68798 interpret the status value returned by *waitpid()* (and *wait()*). The code segment creates a child
 68799 process which does some unspecified work. Meanwhile the parent loops performing calls to
 68800 *waitpid()* to monitor the status of the child. The loop terminates when child termination is
 68801 detected.

```

68802 #include <stdio.h>
68803 #include <stdlib.h>
68804 #include <unistd.h>
68805 #include <sys/wait.h>
68806 ...
68807 pid_t child_pid, wpid;
68808 int status;
68809
68810 child_pid = fork();
68811 if (child_pid == -1) { /* fork() failed */
68812     perror("fork");
68813     exit(EXIT_FAILURE);
68814 }
68815 if (child_pid == 0) { /* This is the child */
68816     /* Child does some work and then terminates */
68817     ...
68818 } else { /* This is the parent */
68819     do {
68820         wpid = waitpid(child_pid, &status, WUNTRACED
68821 #ifdef WCONTINUED /* Not all implementations support this */
68822         | WCONTINUED
68823 #endif
68824     );
68825     if (wpid == -1) {
68826         perror("waitpid");
68827         exit(EXIT_FAILURE);
68828     }
68829     if (WIFEXITED(status)) {
68830         printf("child exited, status=%d\n", WEXITSTATUS(status));
68831     } else if (WIFSIGNALED(status)) {
68832         printf("child killed (signal %d)\n", WTERMSIG(status));
68833     } else if (WIFSTOPPED(status)) {
68834         printf("child stopped (signal %d)\n", WSTOPSIG(status));
68835 #ifdef WIFCONTINUED /* Not all implementations support this */
68836     } else if (WIFCONTINUED(status)) {
68837         printf("child continued\n");

```

```

68837     #endif
68838         } else { /* Non-standard case -- may never happen */
68839             printf("Unexpected status (0x%x)\n", status);
68840         }
68841     } while (!WIFEXITED(status) && !WIFSIGNALED(status));
68842 }

```

68843 **Waiting for a Child Process in a Signal Handler for SIGCHLD**

68844 The following example demonstrates how to use *waitpid()* in a signal handler for SIGCHLD
 68845 without passing -1 as the *pid* argument. (See the APPLICATION USAGE section below for the
 68846 reasons why passing a *pid* of -1 is not recommended.) The method used here relies on the
 68847 standard behavior of *waitpid()* when SIGCHLD is blocked. On historical non-conforming
 68848 systems, the status of some child processes might not be reported.

```

68849 #include <stdlib.h>
68850 #include <stdio.h>
68851 #include <signal.h>
68852 #include <sys/types.h>
68853 #include <sys/wait.h>
68854 #include <unistd.h>
68855
68856 #define CHILDREN 10
68857
68858 static void
68859 handle_sigchld(int signum, siginfo_t *sinfo, void *unused)
68860 {
68861     int status;
68862
68863     /*
68864      * Obtain status information for the child which
68865      * caused the SIGCHLD signal and write its exit code
68866      * to stdout.
68867      */
68868     if (sinfo->si_code != CLD_EXITED)
68869     {
68870         static char msg[] = "wrong si_code\n";
68871         write(2, msg, sizeof msg - 1);
68872     }
68873     else if (waitpid(sinfo->si_pid, &status, 0) == -1)
68874     {
68875         static char msg[] = "waitpid() failed\n";
68876         write(2, msg, sizeof msg - 1);
68877     }
68878     else if (!WIFEXITED(status))
68879     {
68880         static char msg[] = "WIFEXITED was false\n";
68881         write(2, msg, sizeof msg - 1);
68882     }
68883     else
68884     {
68885         int code = WEXITSTATUS(status);
68886         char buf[2];
68887         buf[0] = '0' + code;

```

wait()*System Interfaces*

```

68885         buf[1] = '\n';
68886         write(1, buf, 2);
68887     }
68888 }
68889
68889 int
68890 main(void)
68891 {
68892     int i;
68893     pid_t pid;
68894     struct sigaction sa;
68895
68895     sa.sa_flags = SA_SIGINFO;
68896     sa.sa_sigaction = handle_sigchld;
68897     sigemptyset(&sa.sa_mask);
68898     if (sigaction(SIGCHLD, &sa, NULL) == -1)
68899     {
68900         perror("sigaction");
68901         exit(EXIT_FAILURE);
68902     }
68903
68903     for (i = 0; i < CHILDREN; i++)
68904     {
68905         switch (pid = fork())
68906         {
68907             case -1:
68908                 perror("fork");
68909                 exit(EXIT_FAILURE);
68910             case 0:
68911                 sleep(2);
68912                 _exit(i);
68913         }
68914     }
68915
68915     /* Wait for all the SIGCHLD signals, then terminate on SIGALRM */
68916     alarm(3);
68917     for (;;)
68918         pause();
68919 }

```

APPLICATION USAGE

Calls to *wait()* will collect information about any child process. This may result in interactions with other interfaces that may be waiting for their own children (such as by use of *system()*). For this and other reasons it is recommended that portable applications not use *wait()*, but instead use *waitpid()*. For these same reasons, the use of *waitpid()* with a *pid* argument of -1, and the use of *waitid()* with the *idtype* argument set to P_ALL, are also not recommended for portable applications.

RATIONALE

A call to the *wait()* or *waitpid()* function only returns *status* on an immediate child process of the calling process; that is, a child that was produced by a single *fork()* call (perhaps followed by an *exec* or other function calls) from the parent. If a child produces grandchildren by further use of *fork()*, none of those grandchildren nor any of their descendants affect the behavior of a *wait()* from the original parent process. Nothing in this volume of POSIX.1-2008 prevents an implementation from providing extensions that permit a process to get *status* from a grandchild

or any other process, but a process that does not use such extensions must be guaranteed to see *status* from only its direct children.

The *waitpid()* function is provided for three reasons:

1. To support job control
2. To permit a non-blocking version of the *wait()* function
3. To permit a library routine, such as *system()* or *pclose()*, to wait for its children without interfering with other terminated children for which the process has not waited

The first two of these facilities are based on the *wait3()* function provided by 4.3 BSD. The function uses the *options* argument, which is equivalent to an argument to *wait3()*. The WUNTRACED flag is used only in conjunction with job control on systems supporting job control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped processes in that implementation: processes being traced via the *ptrace()* debugging facility and (untraced) processes stopped by job control signals. Since *ptrace()* is not part of this volume of POSIX.1-2008, only the second type is relevant. The name WUNTRACED was retained because its usage is the same, even though the name is not intuitively meaningful in this context.

The third reason for the *waitpid()* function is to permit independent sections of a process to spawn and wait for children without interfering with each other. For example, the following problem occurs in developing a portable shell, or command interpreter:

```
stream = popen("/bin/true");
(void) system("sleep 100");
(void) pclose(stream);
```

On all historical implementations, the final *pclose()* fails to reap the *wait()* *status* of the *popen()*.

The status values are retrieved by macros, rather than given as specific bit encodings as they are in most historical implementations (and thus expected by existing programs). This was necessary to eliminate a limitation on the number of signals an implementation can support that was inherent in the traditional encodings. This volume of POSIX.1-2008 does require that a *status* value of zero corresponds to a process calling *_exit(0)*, as this is the most common encoding expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

These macros syntactically operate on an arbitrary integer value. The behavior is undefined unless that value is one stored by a successful call to *wait()* or *waitpid()* in the location pointed to by the *stat_loc* argument. An early proposal attempted to make this clearer by specifying each argument as **stat_loc* rather than *stat_val*. However, that did not follow the conventions of other specifications in this volume of POSIX.1-2008 or traditional usage. It also could have implied that the argument to the macro must literally be **stat_loc*; in fact, that value can be stored or passed as an argument to other functions before being interpreted by these macros.

The extension that affects *wait()* and *waitpid()* and is common in historical implementations is the *ptrace()* function. It is called by a child process and causes that child to stop and return a *status* that appears identical to the *status* indicated by WIFSTOPPED. The *status* of *ptrace()* children is traditionally returned regardless of the WUNTRACED flag (or by the *wait()* function). Most applications do not need to concern themselves with such extensions because they have control over what extensions they or their children use. However, applications, such as command interpreters, that invoke arbitrary processes may see this behavior when those arbitrary processes misuse such extensions.

Implementations that support **core** file creation or other implementation-defined actions on termination of some processes traditionally provide a bit in the *status* returned by *wait()* to indicate that such actions have occurred.

wait()*System Interfaces*

Allowing the *wait()* family of functions to discard a pending SIGCHLD signal that is associated with a successfully waited-for child process puts them into the *sigwait()* and *sigwaitinfo()* category with respect to SIGCHLD.

This definition allows implementations to treat a pending SIGCHLD signal as accepted by the process in *wait()*, with the same meaning of “accepted” as when that word is applied to the *sigwait()* family of functions.

Allowing the *wait()* family of functions to behave this way permits an implementation to be able to deal precisely with SIGCHLD signals.

In particular, an implementation that does accept (discard) the SIGCHLD signal can make the following guarantees regardless of the queuing depth of signals in general (the list of waitable children can hold the SIGCHLD queue):

1. If a SIGCHLD signal handler is established via *sigaction()* without the SA_RESETHAND flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal will be delivered to or accepted by the process for every child process that terminates.
2. A single *wait()* issued from a SIGCHLD signal handler can be guaranteed to return immediately with status information for a child process.
3. When SA_SIGINFO is requested, the SIGCHLD signal handler can be guaranteed to receive a non-null pointer to a **siginfo_t** structure that describes a child process for which a wait via *waitpid()* or *waitid()* will not block or fail.
4. The *system()* function will not cause the SIGCHLD handler of a process to be called as a result of the *fork()/exec* executed within *system()* because *system()* will accept the SIGCHLD signal when it performs a *waitpid()* for its child process. This is a desirable behavior of *system()* so that it can be used in a library without causing side-effects to the application linked with the library.

An implementation that does not permit the *wait()* family of functions to accept (discard) a pending SIGCHLD signal associated with a successfully waited-for child, cannot make the guarantees described above for the following reasons:

Guarantee #1

Although it might be assumed that reliable queuing of all SIGCHLD signals generated by the system can make this guarantee, the counter-example is the case of a process that blocks SIGCHLD and performs an indefinite loop of *fork()/wait()* operations. If the implementation supports queued signals, then eventually the system will run out of memory for the queue. The guarantee cannot be made because there must be some limit to the depth of queuing.

Guarantees #2 and #3

These cannot be guaranteed unless the *wait()* family of functions accepts the SIGCHLD signal. Otherwise, a *fork()/wait()* executed while SIGCHLD is blocked (as in the *system()* function) will result in an invocation of the handler when SIGCHLD is unblocked, after the process has disappeared.

Guarantee #4

Although possible to make this guarantee, *system()* would have to set the SIGCHLD handler to SIG_DFL so that the SIGCHLD signal generated by its *fork()* would be discarded (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This would have the undesirable side-effect of discarding all SIGCHLD signals pending to the process.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *exit()*, *fork()*, *system()*, *waitid()*

XBD Section 4.11 (on page 110), *<signal.h>*, *<sys/wait.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The processing of the SIGCHLD signal and the [ECHILD] error is clarified.

The semantics of WIFSTOPPED(*stat_val*), WIFEXITED(*stat_val*), and WIFSIGNALED(*stat_val*) are defined with respect to *posix_spawn()* or *posix_spawnp()* for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/145 is applied, adding the example to the EXAMPLES section.

Issue 7

SD5-XSH-ERN-202 is applied.

APPLICATION USAGE is added, recommending that the *wait()* function not be used.

An additional example for *waitpid()* is added.

waitid()*System Interfaces*69052 **NAME**

69053 waitid — wait for a child process to change state

69054 **SYNOPSIS**

69055 #include <sys/wait.h>

69056 int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);

69057 **DESCRIPTION**

69058 The *waitid()* function shall suspend the calling thread until one child of the process containing
 69059 the calling thread changes state. It records the current state of a child in the structure pointed to
 69060 by *infop*. The fields of the structure pointed to by *infop* are filled in as described for the
 69061 SIGCHLD signal in <signal.h>. If a child process changed state prior to the call to *waitid()*,
 69062 *waitid()* shall return immediately. If more than one thread is suspended in *wait()*, *waitid()*, or
 69063 *waitpid()* waiting for termination of the same process, exactly one thread shall return the process
 69064 status at the time of the target process termination.

69065 The *idtype* and *id* arguments are used to specify which children *waitid()* waits for.

69066 If *idtype* is P_PID, *waitid()* shall wait for the child with a process ID equal to (**pid_t**)*id*.

69067 If *idtype* is P_PGID, *waitid()* shall wait for any child with a process group ID equal to (**pid_t**)*id*.

69068 If *idtype* is P_ALL, *waitid()* shall wait for any children and *id* is ignored.

69069 The *options* argument is used to specify which state changes *waitid()* shall wait for. It is formed
 69070 by OR'ing together the following flags:

69071 WCONTINUED Status shall be returned for any child that was stopped and has been
 69072 continued.

69073 WEXITED Wait for processes that have exited.

69074 WNOHANG Do not hang if no status is available; return immediately.

69075 WNOWAIT Keep the process whose status is returned in *infop* in a waitable state. This
 69076 shall not affect the state of the process; the process may be waited for again
 69077 after this call completes.

69078 WSTOPPED Status shall be returned for any child that has stopped upon receipt of a signal.

69079 Applications shall specify at least one of the flags WEXITED, WSTOPPED, or WCONTINUED to
 69080 be OR'ed in with the *options* argument.

69081 The application shall ensure that the *infop* argument points to a **siginfo_t** structure. If *waitid()*
 69082 returns because a child process was found that satisfied the conditions indicated by the
 69083 arguments *idtype* and *options*, then the structure pointed to by *infop* shall be filled in by the
 69084 system with the status of the process. The *si_signo* member shall always be equal to SIGCHLD.

69085 **RETURN VALUE**

69086 If WNOHANG was specified and status is not available for any process specified by *idtype* and
 69087 *id*, 0 shall be returned. If *waitid()* returns due to the change of state of one of its children, 0 shall
 69088 be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

69089 **ERRORS**

69090 The *waitid()* function shall fail if:

69091 [ECHILD] The calling process has no existing unwaited-for child processes.

69092 [EINTR] The *waitid()* function was interrupted by a signal.

69093 [EINVAL] An invalid value was specified for *options*, or *idtype* and *id* specify an invalid
 69094 set of processes.

EXAMPLES

69095
 69096 None.

APPLICATION USAGE

69098 Calls to *waitid()* with *idtype* equal to P_ALL will collect information about any child process.
 69099 This may result in interactions with other interfaces that may be waiting for their own children
 69100 (such as by use of *system()*). For this reason it is recommended that portable applications not
 69101 use *waitid()* with *idtype* of P_ALL. See also APPLICATION USAGE for *wait()*.

RATIONALE

69102
 69103 None.

FUTURE DIRECTIONS

69104
 69105 None.

SEE ALSO

69107 *exec*, *exit()*, *wait()*

69108 XBD <signal.h>, <sys/wait.h>

CHANGE HISTORY

69109 First released in Issue 4, Version 2.

Issue 5

69112 Moved from X/OPEN UNIX extension to BASE.

69113 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

69115 The normative text is updated to avoid use of the term “must” for application requirements.

Issue 7

69117 Austin Group Interpretation 1003.1-2001 #060 is applied, updating the DESCRIPTION.

69118 The *waitid()* function is moved from the XSI option to the Base.

69119 APPLICATION USAGE is added, recommending that the *waitid()* function not be used with
 69120 *idtype* equal to P_ALL.

69121 The description of the WNOHANG flag is updated.

waitpid()*System Interfaces*69122 **NAME**

69123 waitpid — wait for a child process to stop or terminate

69124 **SYNOPSIS**

69125 #include <sys/wait.h>

69126 pid_t waitpid(pid_t *pid*, int **stat_loc*, int *options*);69127 **DESCRIPTION**69128 Refer to *wait()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***wcpcpy()**69129 **NAME**

69130 wcpcpy — copy a wide-character string, returning a pointer to its end

69131 **SYNOPSIS**69132 CX `#include <wchar.h>`69133 `wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);`69134 **DESCRIPTION**69135 Refer to *wcscpy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

wcpncpy()*System Interfaces*69136 **NAME**

69137 wcpncpy — copy a fixed-size wide-character string, returning a pointer to its end

69138 **SYNOPSIS**

69139 CX #include <wchar.h>

69140 wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,
69141 size_t n);69142 **DESCRIPTION**69143 Refer to *wcsncpy()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

wcrtomb()**NAME**

wcrtomb — convert a wide-character code to a character (restartable)

SYNOPSIS

```
#include <stdio.h>
```

```
size_t wcrtomb(char *restrict s, wchar_t wc, mbstate_t *restrict ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

If *s* is a null pointer, the *wcrtomb()* function shall be equivalent to the call:

```
wcrtomb(buf, L'\0', ps)
```

where *buf* is an internal buffer.

If *s* is not a null pointer, the *wcrtomb()* function shall determine the number of bytes needed to represent the character that corresponds to the wide character given by *wc* (including any shift sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At most {MB_CUR_MAX} bytes are stored. If *wc* is a null wide character, a null byte shall be stored, preceded by any shift sequence needed to restore the initial shift state. The resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *wcrtomb()* function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls *wcrtomb()*.

CX The *wcrtomb()* function need not be thread-safe if called with a NULL *ps* argument.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

RETURN VALUE

The *wcrtomb()* function shall return the number of bytes stored in the array object (including any shift sequences). When *wc* is not a valid wide character, an encoding error shall occur. In this case, the function shall store the value of the macro [EILSEQ] in *errno* and shall return (**size_t**)-1; the conversion state shall be undefined.

ERRORS

The *wcrtomb()* function shall fail if:

[EILSEQ] An invalid wide-character code is detected.

The *wcrtomb()* function may fail if:

[EINVAL] *ps* points to an object that contains an invalid conversion state.

wcrtomb()*System Interfaces*69179 **EXAMPLES**

69180 None.

69181 **APPLICATION USAGE**

69182 None.

69183 **RATIONALE**

69184 None.

69185 **FUTURE DIRECTIONS**

69186 None.

69187 **SEE ALSO**69188 *mbsinit()*, *wcsrtombs()*69189 XBD <*wchar.h*>69190 **CHANGE HISTORY**69191 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
69192 (E).69193 **Issue 6**

69194 In the DESCRIPTION, a note on using this function in a threaded application is added.

69195 Extensions beyond the ISO C standard are marked.

69196 The normative text is updated to avoid use of the term “must” for application requirements.

69197 The *wcrtomb()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.69198 **Issue 7**69199 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *wcrtomb()* function
69200 need not be thread-safe if called with a NULL *ps* argument.

69201 Austin Group Interpretation 1003.1-2001 #170 is applied.

System Interfaces

wscasecmp()**NAME**

`wscasecmp`, `wscasecmp_l`, `wcsncasecmp`, `wcsncasecmp_l` — case-insensitive wide-character string comparison

SYNOPSIS

```
#include <wchar.h>

int wscasecmp(const wchar_t *ws1, const wchar_t *ws2);
int wscasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
    locale_t locale);
int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
    size_t n, locale_t locale);
```

DESCRIPTION

The `wscasecmp()` and `wcsncasecmp()` functions are the wide-character equivalent of the `strcasecmp()` and `strncasecmp()` functions, respectively.

The `wscasecmp()` and `wscasecmp_l()` functions shall compare, while ignoring differences in case, the wide-character string pointed to by `ws1` to the wide-character string pointed to by `ws2`.

The `wcsncasecmp()` and `wcsncasecmp_l()` functions shall compare, while ignoring differences in case, not more than `n` wide-characters from the wide-character string pointed to by `ws1` to the wide-character string pointed to by `ws2`.

When the `LC_CTIME` category of the current locale is from the POSIX locale, these functions shall behave as if the strings had been converted to lowercase and then a byte comparison performed. Otherwise, the results are unspecified.

The information for `wscasecmp_l()` and `wcsncasecmp_l()` about the case of the characters comes from the locale represented by `locale`.

RETURN VALUE

Upon completion, the `wscasecmp()` and `wscasecmp_l()` functions shall return an integer greater than, equal to, or less than 0 if the wide-character string pointed to by `ws1` is, ignoring case, greater than, equal to, or less than the wide-character string pointed to by `ws2`, respectively.

Upon completion, the `wcsncasecmp()` and `wcsncasecmp_l()` functions shall return an integer greater than, equal to, or less than 0 if the possibly null wide-character terminated string pointed to by `ws1` is, ignoring case, greater than, equal to, or less than the possibly null wide-character terminated string pointed to by `ws2`, respectively.

No return values are reserved to indicate an error.

ERRORS

The `wscasecmp_l()` and `wcsncasecmp_l()` functions may fail if:

[EINVAL] `locale` is not a valid locale object handle.

wcscasecmp()*System Interfaces*69238 **EXAMPLES**

69239 None.

69240 **APPLICATION USAGE**

69241 None.

69242 **RATIONALE**

69243 None.

69244 **FUTURE DIRECTIONS**

69245 None.

69246 **SEE ALSO**69247 [*strcasecmp\(\)*](#), [*wcscmp\(\)*](#), [*wcsncmp\(\)*](#)69248 XBD [**<wchar.h>**](#)69249 **CHANGE HISTORY**

69250 First released in Issue 7.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

wscat — concatenate two wide-character strings

SYNOPSIS

```
#include <wchar.h>
```

```
wchar_t *wscat(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *wscat()* function shall append a copy of the wide-character string pointed to by *ws2* (including the terminating null wide-character code) to the end of the wide-character string pointed to by *ws1*. The initial wide-character code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *wscat()* function shall return *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[*wcsncat\(\)*](#)

XBD [*<wchar.h>*](#)

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The Open Group Corrigendum U040/2 is applied. In the RETURN VALUE section, *s1* is changed to *ws1*.

The *wscat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

wcschr()*System Interfaces*69286 **NAME**69287 **wcschr** — wide-character string scanning operation69288 **SYNOPSIS**69289 `#include <wchar.h>`69290 `wchar_t *wcschr(const wchar_t *ws, wchar_t wc);`69291 **DESCRIPTION**

69292 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 69293 conflict between the requirements described here and the ISO C standard is unintentional. This
 69294 volume of POSIX.1-2008 defers to the ISO C standard.

69295 The *wcschr()* function shall locate the first occurrence of *wc* in the wide-character string pointed
 69296 to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type
 69297 **wchar_t** and a wide-character code corresponding to a valid character in the current locale. The
 69298 terminating null wide-character code is considered to be part of the wide-character string.

69299 **RETURN VALUE**

69300 Upon completion, *wcschr()* shall return a pointer to the wide-character code, or a null pointer if
 69301 the wide-character code is not found.

69302 **ERRORS**

69303 No errors are defined.

69304 **EXAMPLES**

69305 None.

69306 **APPLICATION USAGE**

69307 None.

69308 **RATIONALE**

69309 None.

69310 **FUTURE DIRECTIONS**

69311 None.

69312 **SEE ALSO**69313 [*wcsrchr\(\)*](#)69314 XBD [**<wchar.h>**](#)69315 **CHANGE HISTORY**

69316 First released in Issue 4. Derived from the MSE working draft.

69317 **Issue 6**

69318 The normative text is updated to avoid use of the term “must” for application requirements.

NAME

wcscmp — compare two wide-character strings

SYNOPSIS

```
#include <wchar.h>
```

```
int wcscmp(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *wcscmp()* function shall compare the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared.

RETURN VALUE

Upon completion, *wcscmp()* shall return an integer greater than, equal to, or less than 0, if the wide-character string pointed to by *ws1* is greater than, equal to, or less than the wide-character string pointed to by *ws2*, respectively.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscasecmp(), *wcsncmp()*

XBD [<wchar.h>](#)

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

wscoll()*System Interfaces*69351 **NAME**69352 `wscoll`, `wscoll_l` — wide-character string comparison using collating information69353 **SYNOPSIS**69354 `#include <wchar.h>`69355 `int wscoll(const wchar_t *ws1, const wchar_t *ws2);`69356 CX `int wscoll_l(const wchar_t *ws1, const wchar_t *ws2,`
69357 `locale_t locale);`69358 **DESCRIPTION**69359 CX For `wscoll()`: The functionality described on this reference page is aligned with the ISO C
69360 standard. Any conflict between the requirements described here and the ISO C standard is
69361 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.69362 CX The `wscoll()` and `wscoll_l()` functions shall compare the wide-character string pointed to by
69363 `ws1` to the wide-character string pointed to by `ws2`, both interpreted as appropriate to the
69364 CX `LC_COLLATE` category of the current locale of the process, or the locale represented by `locale`,
69365 respectively.69366 CX The `wscoll()` and `wscoll_l()` functions shall not change the setting of `errno` if successful.69367 CX An application wishing to check for error situations should set `errno` to 0 before calling `wscoll()`
69368 or `wscoll_l()`. If `errno` is non-zero on return, an error has occurred.69369 **RETURN VALUE**69370 CX Upon successful completion, `wscoll()` and `wscoll_l()` shall return an integer greater than, equal
69371 to, or less than 0, according to whether the wide-character string pointed to by `ws1` is greater
69372 than, equal to, or less than the wide-character string pointed to by `ws2`, when both are
69373 CX interpreted as appropriate to the current locale, or to the locale represented by `locale`,
69374 CX respectively. On error, `wscoll()` and `wscoll_l()` shall set `errno`, but no return value is reserved
69375 to indicate an error.69376 **ERRORS**

69377 These functions may fail if:

69378 CX [EINVAL] The `ws1` or `ws2` arguments contain wide-character codes outside the domain of
69379 the collating sequence.69380 The `wscoll_l()` function may fail if:69381 CX [EINVAL] `locale` is not a valid locale object handle.69382 **EXAMPLES**

69383 None.

69384 **APPLICATION USAGE**69385 The `wcsxfrm()` and `wcscmp()` functions should be used for sorting large lists.69386 **RATIONALE**

69387 None.

69388 **FUTURE DIRECTIONS**

69389 None.

69390 **SEE ALSO**69391 `wcscmp()`, `wcsxfrm()`69392 XBD `<wchar.h>`

69393 **CHANGE HISTORY**

69394 First released in Issue 4. Derived from the MSE working draft.

69395 **Issue 5**

69396 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

69397 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.69398 **Issue 7**69399 The *wscoll_l()* function is added from The Open Group Technical Standard, 2006, Extended API
69400 Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

wcscpy()*System Interfaces*69401 **NAME**69402 `wcpcpy`, `wcscpy` — copy a wide-character string, returning a pointer to its end69403 **SYNOPSIS**69404 `#include <wchar.h>`

```
69405 CX wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);
69406 wchar_t *wcscpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

69407 **DESCRIPTION**

69408 CX For `wcscpy()`: The functionality described on this reference page is aligned with the ISO C
 69409 standard. Any conflict between the requirements described here and the ISO C standard is
 69410 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

69411 CX The `wcpcpy()` and `wcscpy()` functions shall copy the wide-character string pointed to by `ws2`
 69412 (including the terminating null wide-character code) into the array pointed to by `ws1`.

69413 The application shall ensure that there is room for at least `wcslen(ws2)+1` wide characters in the
 69414 `ws1` array, and that the `ws2` and `ws1` arrays do not overlap.

69415 If copying takes place between objects that overlap, the behavior is undefined.

69416 **RETURN VALUE**

69417 CX The `wcpcpy()` function shall return a pointer to the terminating null wide-character code copied
 69418 into the `ws1` buffer.

69419 The `wcscpy()` function shall return `ws1`.

69420 No return values are reserved to indicate an error.

69421 **ERRORS**

69422 No errors are defined.

69423 **EXAMPLES**

69424 None.

69425 **APPLICATION USAGE**

69426 None.

69427 **RATIONALE**

69428 None.

69429 **FUTURE DIRECTIONS**

69430 None.

69431 **SEE ALSO**

69432 `strcpy()`, `wcsdup()`, `wcsncpy()`

69433 XBD `<wchar.h>`

69434 **CHANGE HISTORY**

69435 First released in Issue 4. Derived from the MSE working draft.

69436 **Issue 6**

69437 The `wcscpy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

69438 **Issue 7**

69439 The `wcpcpy()` function is added from The Open Group Technical Standard, 2006, Extended API
 69440 Set Part 1.

69441 **NAME**

69442 wcscspn — get the length of a complementary wide substring

69443 **SYNOPSIS**

69444 #include <wchar.h>

69445 size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);

69446 **DESCRIPTION**

69447 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69448 conflict between the requirements described here and the ISO C standard is unintentional. This
 69449 volume of POSIX.1-2008 defers to the ISO C standard.

69450 The *wcscspn()* function shall compute the length (in wide characters) of the maximum initial
 69451 segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character
 69452 codes *not* from the wide-character string pointed to by *ws2*.

69453 **RETURN VALUE**

69454 The *wcscspn()* function shall return the length of the initial substring of *ws1*; no return value is
 69455 reserved to indicate an error.

69456 **ERRORS**

69457 No errors are defined.

69458 **EXAMPLES**

69459 None.

69460 **APPLICATION USAGE**

69461 None.

69462 **RATIONALE**

69463 None.

69464 **FUTURE DIRECTIONS**

69465 None.

69466 **SEE ALSO**69467 *wcsspn()*

69468 XBD <wchar.h>

69469 **CHANGE HISTORY**

69470 First released in Issue 4. Derived from the MSE working draft.

69471 **Issue 5**

69472 The RETURN VALUE section is updated to indicate that *wcscspn()* returns the length of *ws1*,
 69473 rather than *ws1* itself.

wcsdup()

System Interfaces

69474 **NAME**69475 **wcsdup** — duplicate a wide-character string69476 **SYNOPSIS**

```
69477 CX    #include <wchar.h>
69478    wchar_t *wcsdup(const wchar_t *string);
```

69479 **DESCRIPTION**69480 The *wcsdup()* function is the wide-character equivalent of the *strdup()* function.

69481 The *wcsdup()* function shall return a pointer to a new wide-character string, allocated as if by a
 69482 call to *malloc()*, which is the duplicate of the wide-character string *string*. The returned pointer
 69483 can be passed to *free()*. A null pointer is returned if the new wide-character string cannot be
 69484 created.

69485 **RETURN VALUE**

69486 Upon successful completion, the *wcsdup()* function shall return a pointer to the newly allocated
 69487 wide-character string. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

69488 **ERRORS**69489 The *wcsdup()* function shall fail if:

69490 [ENOMEM] Memory large enough for the duplicate string could not be allocated.

69491 **EXAMPLES**

69492 None.

69493 **APPLICATION USAGE**

69494 For functions that allocate memory as if by *malloc()*, the application should release such memory
 69495 when it is no longer required by a call to *free()*. For *wcsdup()*, this is the return value.

69496 **RATIONALE**

69497 None.

69498 **FUTURE DIRECTIONS**

69499 None.

69500 **SEE ALSO**69501 *free()*, *strdup()*, *wscpy()*69502 XBD **<wchar.h>**69503 **CHANGE HISTORY**

69504 First released in Issue 7.

NAME

wcsftime — convert date and time to a wide-character string

SYNOPSIS

```
#include <wchar.h>
```

```
size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,  
const wchar_t *restrict format, const struct tm *restrict timeptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *wcsftime()* function shall be equivalent to the *strftime()* function, except that:

- The argument *wcs* points to the initial element of an array of wide characters into which the generated output is to be placed.
- The argument *maxsize* indicates the maximum number of wide characters to be placed in the output array.
- The argument *format* is a wide-character string and the conversion specifications are replaced by corresponding sequences of wide characters.
- The return value indicates the number of wide characters placed in the output array.

If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

If the total number of resulting wide-character codes including the terminating null wide-character code is no more than *maxsize*, *wcsftime()* shall return the number of wide-character codes placed into the array pointed to by *wcs*, not including the terminating null wide-character code. Otherwise, zero is returned and the contents of the array are unspecified.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strftime()

XBD *<wchar.h>*

CHANGE HISTORY

First released in Issue 4.

wcsftime()*System Interfaces*69544 **Issue 5**

69545 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

69546 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of the *format*
69547 argument is changed from **const char *** to **const wchar_t ***.69548 **Issue 6**69549 The *wcsftime()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

wcslen, wcsnlen — get length of a fixed-sized wide-character string

SYNOPSIS

```
#include <wchar.h>
```

```
size_t wcslen(const wchar_t *ws);
```

```
CX size_t wcsnlen(const wchar_t *ws, size_t maxlen);
```

DESCRIPTION

CX For *wcslen()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *wcslen()* function shall compute the number of wide-character codes in the wide-character string to which *ws* points, not including the terminating null wide-character code.

CX The *wcsnlen()* function shall compute the smaller of the number of wide characters in the string to which *ws* points, not including the terminating null wide-character code, and the value of *maxlen*. The *wcsnlen()* function shall never examine more than the first *maxlen* characters of the wide-character string pointed to by *ws*.

RETURN VALUE

The *wcslen()* function shall return the length of *ws*.

CX The *wcsnlen()* function shall return an integer containing the smaller of either the length of the wide-character string pointed to by *ws* or *maxlen*.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[strlen\(\)](#)

XBD [<wchar.h>](#)

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 7

The *wcsnlen()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

wcsncasecmp()*System Interfaces*69589 **NAME**69590 `wcsncasecmp`, `wcsncasecmp_l` — case-insensitive wide-character string comparison69591 **SYNOPSIS**

```
69592 CX    #include <wchar.h>
69593
69593    int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
69594    int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
69595                      size_t n, locale_t locale);
```

69596 **DESCRIPTION**69597 Refer to *wcscasecmp()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

wcsncat — concatenate a wide-character string with part of another

SYNOPSIS

```
#include <wchar.h>
```

```
wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict ws2,  
size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *wcsncat()* function shall append not more than *n* wide-character codes (a null wide-character code and wide-character codes that follow it are not appended) from the array pointed to by *ws2* to the end of the wide-character string pointed to by *ws1*. The initial wide-character code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. A terminating null wide-character code shall always be appended to the result. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *wcsncat()* function shall return *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[*wcscat\(\)*](#)

XBD [*<wchar.h>*](#)

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The *wcsncat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

wcsncmp()

System Interfaces

69633 **NAME**69634 `wcsncmp` — compare part of two wide-character strings69635 **SYNOPSIS**69636 `#include <wchar.h>`69637 `int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);`69638 **DESCRIPTION**

69639 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69640 conflict between the requirements described here and the ISO C standard is unintentional. This
 69641 volume of POSIX.1-2008 defers to the ISO C standard.

69642 The `wcsncmp()` function shall compare not more than *n* wide-character codes (wide-character
 69643 codes that follow a null wide-character code are not compared) from the array pointed to by *ws1*
 69644 to the array pointed to by *ws2*.

69645 The sign of a non-zero return value shall be determined by the sign of the difference between the
 69646 values of the first pair of wide-character codes that differ in the objects being compared.

69647 **RETURN VALUE**

69648 Upon successful completion, `wcsncmp()` shall return an integer greater than, equal to, or less
 69649 than 0, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less
 69650 than the possibly null-terminated array pointed to by *ws2*, respectively.

69651 **ERRORS**

69652 No errors are defined.

69653 **EXAMPLES**

69654 None.

69655 **APPLICATION USAGE**

69656 None.

69657 **RATIONALE**

69658 None.

69659 **FUTURE DIRECTIONS**

69660 None.

69661 **SEE ALSO**69662 `wscasecmp()`, `wscmp()`69663 XBD `<wchar.h>`69664 **CHANGE HISTORY**

69665 First released in Issue 4. Derived from the MSE working draft.

NAME

wcpncpy, wcsncpy — copy a fixed-size wide-character string, returning a pointer to its end

SYNOPSIS

```
#include <wchar.h>
```

```
CX  wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,
    size_t n);
    wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
```

DESCRIPTION

CX For *wcsncpy()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

CX The *wcpncpy()* and *wcsncpy()* functions shall copy not more than *n* wide-character codes (wide-character codes that follow a null wide-character code are not copied) from the array pointed to by *ws2* to the array pointed to by *ws1*. If copying takes place between objects that overlap, the behavior is undefined.

If the array pointed to by *ws2* is a wide-character string that is shorter than *n* wide-character codes, null wide-character codes shall be appended to the copy in the array pointed to by *ws1*, until *n* wide-character codes in all are written.

RETURN VALUE

CX If any null wide-character codes were written into the destination, the *wcpncpy()* function shall return the address of the first such null wide-character code. Otherwise, it shall return *&ws1[n]*.

The *wcsncpy()* function shall return *ws1*.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

If there is no null wide-character code in the first *n* wide-character codes of the array pointed to by *ws2*, the result is not null-terminated.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncpy(), *wcscpy()*

XBD *<wchar.h>*

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

wcsncpy()*System Interfaces*69706 **Issue 6**69707 The *wcsncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.69708 **Issue 7**69709 The *wcpncpy()* function is added from The Open Group Technical Standard, 2006, Extended API
69710 Set Part 1.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*System Interfaces***wcsnlen()**69711 **NAME**

69712 wcsnlen — get length of a fixed-sized wide-character string

69713 **SYNOPSIS**

69714 CX #include <wchar.h>

69715 size_t wcsnlen(const wchar_t *ws, size_t maxlen);

69716 **DESCRIPTION**69717 Refer to *wcslen()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

wcsnrtombs()*System Interfaces*69718 **NAME**

69719 wcsnrtombs — convert wide-character string to multi-byte string

69720 **SYNOPSIS**

69721 CX #include <wchar.h>

```
69722       size_t wcsnrtombs(char *restrict dst, const wchar_t **restrict src,  
69723           size_t nwc, size_t len, mbstate_t *restrict ps);
```

69724 **DESCRIPTION**69725 Refer to *wcsrtombs()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

69726 **NAME**69727 `wcsprk` — scan a wide-character string for a wide-character code69728 **SYNOPSIS**69729 `#include <wchar.h>`69730 `wchar_t *wcsprk(const wchar_t *ws1, const wchar_t *ws2);`69731 **DESCRIPTION**

69732 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69733 conflict between the requirements described here and the ISO C standard is unintentional. This
 69734 volume of POSIX.1-2008 defers to the ISO C standard.

69735 The `wcsprk()` function shall locate the first occurrence in the wide-character string pointed to by
 69736 `ws1` of any wide-character code from the wide-character string pointed to by `ws2`.

69737 **RETURN VALUE**

69738 Upon successful completion, `wcsprk()` shall return a pointer to the wide-character code or a null
 69739 pointer if no wide-character code from `ws2` occurs in `ws1`.

69740 **ERRORS**

69741 No errors are defined.

69742 **EXAMPLES**

69743 None.

69744 **APPLICATION USAGE**

69745 None.

69746 **RATIONALE**

69747 None.

69748 **FUTURE DIRECTIONS**

69749 None.

69750 **SEE ALSO**69751 `wcschr()`, `wcsrchr()`69752 XBD `<wchar.h>`69753 **CHANGE HISTORY**

69754 First released in Issue 4. Derived from the MSE working draft.

wcsrchr()*System Interfaces*69755 **NAME**

69756 wcsrchr — wide-character string scanning operation

69757 **SYNOPSIS**

69758 #include <wchar.h>

69759 wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

69760 **DESCRIPTION**

69761 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69762 conflict between the requirements described here and the ISO C standard is unintentional. This
 69763 volume of POSIX.1-2008 defers to the ISO C standard.

69764 The *wcsrchr()* function shall locate the last occurrence of *wc* in the wide-character string pointed
 69765 to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type
 69766 **wchar_t** and a wide-character code corresponding to a valid character in the current locale. The
 69767 terminating null wide-character code shall be considered to be part of the wide-character string.

69768 **RETURN VALUE**

69769 Upon successful completion, *wcsrchr()* shall return a pointer to the wide-character code or a null
 69770 pointer if *wc* does not occur in the wide-character string.

69771 **ERRORS**

69772 No errors are defined.

69773 **EXAMPLES**

69774 None.

69775 **APPLICATION USAGE**

69776 None.

69777 **RATIONALE**

69778 None.

69779 **FUTURE DIRECTIONS**

69780 None.

69781 **SEE ALSO**69782 *wcschr()*

69783 XBD <wchar.h>

69784 **CHANGE HISTORY**

69785 First released in Issue 4. Derived from the MSE working draft.

69786 **Issue 6**

69787 The normative text is updated to avoid use of the term “must” for application requirements.

NAME

`wcsnrtombs`, `wcsrtombs` — convert a wide-character string to a character string (restartable)

SYNOPSIS

```
#include <wchar.h>
```

```
size_t wcsnrtombs(char *restrict dst, const wchar_t **restrict src,
size_t nwc, size_t len, mbstate_t *restrict ps);
size_t wcsrtombs(char *restrict dst, const wchar_t **restrict src,
size_t len, mbstate_t *restrict ps);
```

DESCRIPTION

For `wcsrtombs()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The `wcsrtombs()` function shall convert a sequence of wide characters from the array indirectly pointed to by `src` into a sequence of corresponding characters, beginning in the conversion state described by the object pointed to by `ps`. If `dst` is not a null pointer, the converted characters shall then be stored into the array pointed to by `dst`. Conversion continues up to and including a terminating null wide character, which shall also be stored. Conversion shall stop earlier in the following cases:

- When a code is reached that does not correspond to a valid character
- When the next character would exceed the limit of `len` total bytes to be stored in the array pointed to by `dst` (and `dst` is not a null pointer)

Each conversion shall take place as if by a call to the `wcrtomb()` function.

If `dst` is not a null pointer, the pointer object pointed to by `src` shall be assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted (if any). If conversion stopped due to reaching a terminating null wide character, the resulting state described shall be the initial conversion state.

If `ps` is a null pointer, the `wcsrtombs()` function shall use its own internal `mbstate_t` object, which is initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t` object pointed to by `ps` shall be used to completely describe the current conversion state of the associated character sequence.

The `wcsrtombs()` function need not be thread-safe if called with a NULL `ps` argument.

The `wcsnrtombs()` function shall be equivalent to the `wcsrtombs()` function, except that the conversion is limited to the first `nwc` wide characters.

The behavior of these functions shall be affected by the `LC_CTYPE` category of the current locale.

The implementation shall behave as if no function defined in System Interfaces volume of POSIX.1-2008 calls these functions.

RETURN VALUE

If conversion stops because a code is reached that does not correspond to a valid character, an encoding error occurs. In this case, these functions shall store the value of the macro `[EILSEQ]` in `errno` and return `(size_t)-1`; the conversion state is undefined. Otherwise, these functions shall return the number of bytes in the resulting character sequence, not including the terminating null (if any).

wcsrtombs()*System Interfaces***69830 ERRORS**

69831 These functions shall fail if:

69832 [EILSEQ] A wide-character code does not correspond to a valid character.

69833 These functions may fail if:

69834 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

69835 EXAMPLES

69836 None.

69837 APPLICATION USAGE

69838 None.

69839 RATIONALE

69840 None.

69841 FUTURE DIRECTIONS

69842 None.

69843 SEE ALSO

69844 *mbsinit()*, *wcrtomb()*

69845 XBD <wchar.h>

69846 CHANGE HISTORY

69847 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
69848 (E).

69849 Issue 6

69850 In the DESCRIPTION, a note on using this function in a threaded application is added.

69851 Extensions beyond the ISO C standard are marked.

69852 The normative text is updated to avoid use of the term “must” for application requirements.

69853 The *wcsrtombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

69854 Issue 7

69855 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *wcsrtombs()* function
69856 need not be thread-safe if called with a NULL *ps* argument.

69857 Austin Group Interpretation 1003.1-2001 #170 is applied.

69858 The *wcnsrtombs()* function is added from The Open Group Technical Standard, 2006, Extended
69859 API Set Part 1.

NAME

wcsspn — get the length of a wide substring

SYNOPSIS

```
#include <wchar.h>
```

```
size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *wcsspn()* function shall compute the length (in wide characters) of the maximum initial segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character codes from the wide-character string pointed to by *ws2*.

RETURN VALUE

The *wcsspn()* function shall return the length of the initial substring of *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[*wcscspn\(\)*](#)

XBD [*<wchar.h>*](#)

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The RETURN VALUE section is updated to indicate that *wcsspn()* returns the length of *ws1* rather than *ws1* itself.

wcsstr()*System Interfaces*69893 **NAME**69894 **wcsstr** — find a wide-character substring69895 **SYNOPSIS**69896 `#include <wchar.h>`69897 `wchar_t *wcsstr(const wchar_t *restrict ws1,`
69898 `const wchar_t *restrict ws2);`69899 **DESCRIPTION**69900 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
69901 conflict between the requirements described here and the ISO C standard is unintentional. This
69902 volume of POSIX.1-2008 defers to the ISO C standard.69903 The *wcsstr()* function shall locate the first occurrence in the wide-character string pointed to by
69904 *ws1* of the sequence of wide characters (excluding the terminating null wide character) in the
69905 wide-character string pointed to by *ws2*.69906 **RETURN VALUE**69907 Upon successful completion, *wcsstr()* shall return a pointer to the located wide-character string,
69908 or a null pointer if the wide-character string is not found.69909 If *ws2* points to a wide-character string with zero length, the function shall return *ws1*.69910 **ERRORS**

69911 No errors are defined.

69912 **EXAMPLES**

69913 None.

69914 **APPLICATION USAGE**

69915 None.

69916 **RATIONALE**

69917 None.

69918 **FUTURE DIRECTIONS**

69919 None.

69920 **SEE ALSO**69921 [*wcschr\(\)*](#)69922 XBD [*<wchar.h>*](#)69923 **CHANGE HISTORY**69924 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
69925 (E).69926 **Issue 6**69927 The *wcsstr()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

NAME

wcstod, wcstof, wcstold — convert a wide-character string to a double-precision number

SYNOPSIS

```
#include <wchar.h>

double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);
float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);
long double wcstold(const wchar_t *restrict nptr,
    wchar_t **restrict endptr);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to **double**, **float**, and **long double** representation, respectively. First, they shall decompose the input wide-character string into three parts:

1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by *iswspace()*)
2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
3. A final wide-character string of one or more unrecognized wide-character codes, including the terminating null wide-character code of the input wide-character string

Then they shall attempt to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional '+' or '-' sign, then one of the following:

- A non-empty sequence of decimal digits optionally containing a radix character; then an optional exponent part consisting of the wide character 'e' or the wide character 'E', optionally followed by a '+' or '-' wide character, and then followed by one or more decimal digits
- A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix character; then an optional binary exponent part consisting of the wide character 'p' or the wide character 'P', optionally followed by a '+' or '-' wide character, and then followed by one or more decimal digits
- One of INF or INFINITY, or any other wide string equivalent except for case
- One of NAN or NAN(*n-wchar-sequence_{opt}*), or any other wide string ignoring case in the NAN part, where:

```
n-wchar-sequence:
    digit
    nondigit
    n-wchar-sequence digit
    n-wchar-sequence nondigit
```

The subject sequence is defined as the longest initial subsequence of the input wide string, starting with the first non-white-space wide character, that is of the expected form. The subject sequence contains no wide characters if the input wide string is not of the expected form.

wcstod()*System Interfaces*

- 69972 If the subject sequence has the expected form for a floating-point number, the sequence of wide
 69973 characters starting with the first digit or the radix character (whichever occurs first) shall be
 69974 interpreted as a floating constant according to the rules of the C language, except that the radix
 69975 character shall be used in place of a period, and that if neither an exponent part nor a radix
 69976 character appears in a decimal floating-point number, or if a binary exponent part does not
 69977 appear in a hexadecimal floating-point number, an exponent part of the appropriate type with
 69978 value zero shall be assumed to follow the last digit in the string. If the subject sequence begins
 69979 with a minus-sign, the sequence shall be interpreted as negated. A wide-character sequence INF
 69980 or INFINITY shall be interpreted as an infinity, if representable in the return type, else as if it
 69981 were a floating constant that is too large for the range of the return type. A wide-character
 69982 sequence NAN or NAN(*n-wchar-sequence_{opt}*) shall be interpreted as a quiet NaN, if supported in
 69983 the return type, else as if it were a subject sequence part that does not have the expected form;
 69984 the meaning of the *n-wchar* sequences is implementation-defined. A pointer to the final wide
 69985 string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.
- 69986 If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the
 69987 conversion shall be rounded in an implementation-defined manner.
- 69988 CX The radix character shall be as defined in the locale of the process (category LC_NUMERIC). In
 69989 the POSIX locale, or in a locale where the radix character is not defined, the radix character shall
 69990 default to a <period> (' . ').
- 69991 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
 69992 accepted.
- 69993 If the subject sequence is empty or does not have the expected form, no conversion shall be
 69994 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
 69995 *endptr* is not a null pointer.
- 69996 CX The *wcstod()* function shall not change the setting of *errno* if successful.
- 69997 Since 0 is returned on error and is also a valid return on success, an application wishing to check
 69998 for error situations should set *errno* to 0, then call *wcstod()*, *wcstof()*, or *wcstold()*, then check
 69999 *errno*.
- 70000 **RETURN VALUE**
- 70001 Upon successful completion, these functions shall return the converted value. If no conversion
 70002 CX could be performed, 0 shall be returned and *errno* may be set to [EINVAL].
- 70003 If the correct value is outside the range of representable values, ±HUGE_VAL, ±HUGE_VALF, or
 70004 ±HUGE_VALL shall be returned (according to the sign of the value), and *errno* shall be set to
 70005 [ERANGE].
- 70006 If the correct value would cause underflow, a value whose magnitude is no greater than the
 70007 smallest normalized positive number in the return type shall be returned and *errno* set to
 70008 [ERANGE].
- 70009 **ERRORS**
- 70010 The *wcstod()* function shall fail if:
- 70011 [ERANGE] The value to be returned would cause overflow or underflow.
- 70012 The *wcstod()* function may fail if:
- 70013 CX [EINVAL] No conversion could be performed.

EXAMPLES

None.

APPLICATION USAGE

If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the result is not exactly representable, the result should be one of the two numbers in the appropriate internal format that are adjacent to the hexadecimal floating source value, with the extra stipulation that the error should have a correct sign for the current rounding direction.

If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in `<float.h>`) significant digits, the result should be correctly rounded. If the subject sequence *D* has the decimal form and more than DECIMAL_DIG significant digits, consider the two bounding, adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the values of *L*, *D*, and *U* satisfy "*L* ≤ *D* ≤ *U*". The result should be one of the (equal or adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current rounding direction, with the extra stipulation that the error with respect to *D* should have a correct sign for the current rounding direction.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`fscanf()`, `iswspace()`, `localeconv()`, `setlocale()`, `wcstol()`

XBD Chapter 7 (on page 135), `<float.h>`, `<wchar.h>`

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The `wcstod()` prototype is updated.
- The `wcstof()` and `wcstold()` functions are added.
- If the correct value for `wcstod()` would cause underflow, the return value changed from 0 (as specified in Issue 5) to the smallest normalized positive number.
- The DESCRIPTION, RETURN VALUE, and APPLICATION USAGE sections are extensively updated.

ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/66 is applied, correcting the second paragraph in the RETURN VALUE section.

wcstod()

System Interfaces

70056 **Issue 7**
70057

Austin Group Interpretation 1003.1-2001 #015 is applied.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

wcstoimax, wcstoumax — convert a wide-character string to an integer type

SYNOPSIS

```
#include <stddef.h>
#include <inttypes.h>

intmax_t wcstoimax(const wchar_t *restrict nptr,
                   wchar_t **restrict endptr, int base);
uintmax_t wcstoumax(const wchar_t *restrict nptr,
                    wchar_t **restrict endptr, int base);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

These functions shall be equivalent to the *wcstol()*, *wcstoll()*, *wcstoul()*, and *wcstoull()* functions, respectively, except that the initial portion of the wide string shall be converted to **intmax_t** and **uintmax_t** representation, respectively.

RETURN VALUE

These functions shall return the converted value, if any.

If no conversion could be performed, zero shall be returned. If the correct value is outside the range of representable values, {INTMAX_MAX}, {INTMAX_MIN}, or {UINTMAX_MAX} shall be returned (according to the return type and sign of the value, if any), and *errno* shall be set to [ERANGE].

ERRORS

These functions shall fail if:

- [EINVAL] The value of *base* is not supported.
- [ERANGE] The value to be returned is not representable.

These functions may fail if:

- [EINVAL] No conversion could be performed.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcstol(), *wcstoul()*

XBD *<inttypes.h>*, *<stddef.h>*

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

wcstok()*System Interfaces*70099 **NAME**70100 `wcstok` — split a wide-character string into tokens70101 **SYNOPSIS**70102 `#include <wchar.h>`

```
70103     wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2,
70104                     wchar_t **restrict ptr);
```

70105 **DESCRIPTION**

70106 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 70107 conflict between the requirements described here and the ISO C standard is unintentional. This
 70108 volume of POSIX.1-2008 defers to the ISO C standard.

70109 A sequence of calls to `wcstok()` shall break the wide-character string pointed to by `ws1` into a
 70110 sequence of tokens, each of which shall be delimited by a wide-character code from the wide-
 70111 character string pointed to by `ws2`. The `ptr` argument points to a caller-provided **wchar_t** pointer
 70112 into which the `wcstok()` function shall store information necessary for it to continue scanning the
 70113 same wide-character string.

70114 The first call in the sequence has `ws1` as its first argument, and is followed by calls with a null
 70115 pointer as their first argument. The separator string pointed to by `ws2` may be different from call
 70116 to call.

70117 The first call in the sequence shall search the wide-character string pointed to by `ws1` for the first
 70118 wide-character code that is *not* contained in the current separator string pointed to by `ws2`. If no
 70119 such wide-character code is found, then there are no tokens in the wide-character string pointed
 70120 to by `ws1` and `wcstok()` shall return a null pointer. If such a wide-character code is found, it shall
 70121 be the start of the first token.

70122 The `wcstok()` function shall then search from there for a wide-character code that *is* contained in
 70123 the current separator string. If no such wide-character code is found, the current token extends
 70124 to the end of the wide-character string pointed to by `ws1`, and subsequent searches for a token
 70125 shall return a null pointer. If such a wide-character code is found, it shall be overwritten by a
 70126 null wide character, which terminates the current token. The `wcstok()` function shall save a
 70127 pointer to the following wide-character code, from which the next search for a token shall start.

70128 Each subsequent call, with a null pointer as the value of the first argument, shall start searching
 70129 from the saved pointer and behave as described above.

70130 The implementation shall behave as if no function calls `wcstok()`.

70131 **RETURN VALUE**

70132 Upon successful completion, the `wcstok()` function shall return a pointer to the first wide-
 70133 character code of a token. Otherwise, if there is no token, `wcstok()` shall return a null pointer.

70134 **ERRORS**

70135 No errors are defined.

70136 **EXAMPLES**

70137 None.

70138 **APPLICATION USAGE**

70139 None.

70140 **RATIONALE**

70141 None.

70142 **FUTURE DIRECTIONS**

70143 None.

70144 **SEE ALSO**70145 XBD [<wchar.h>](#)70146 **CHANGE HISTORY**

70147 First released in Issue 4.

70148 **Issue 5**70149 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, a third argument is
70150 added to the definition of *wcstok()* in the SYNOPSIS.70151 **Issue 6**70152 The *wcstok()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

wcstol()*System Interfaces*70153 **NAME**70154 **wcstol, wcstoll** — convert a wide-character string to a long integer70155 **SYNOPSIS**

```
70156 #include <wchar.h>
70157 long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr,
70158             int base);
70159 long long wcstoll(const wchar_t *restrict nptr,
70160                  wchar_t **restrict endptr, int base);
```

70161 **DESCRIPTION**

70162 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 70163 conflict between the requirements described here and the ISO C standard is unintentional. This
 70164 volume of POSIX.1-2008 defers to the ISO C standard.

70165 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to
 70166 **long** and **long long**, respectively. First, they shall decompose the input string into three parts:

- 70167 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by
 70168 *iswspace()*)
- 70169 2. A subject sequence interpreted as an integer represented in some radix determined by the
 70170 value of *base*
- 70171 3. A final wide-character string of one or more unrecognized wide-character codes,
 70172 including the terminating null wide-character code of the input wide-character string

70173 Then they shall attempt to convert the subject sequence to an integer, and return the result.

70174 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,
 70175 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal
 70176 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal
 70177 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'
 70178 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
 70179 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

70180 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
 70181 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
 70182 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'
 70183 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less
 70184 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character code
 70185 representations of 0x or 0X may optionally precede the sequence of letters and digits, following
 70186 the sign if present.

70187 The subject sequence is defined as the longest initial subsequence of the input wide-character
 70188 string, starting with the first non-white-space wide-character code that is of the expected form.
 70189 The subject sequence contains no wide-character codes if the input wide-character string is
 70190 empty or consists entirely of white-space wide-character code, or if the first non-white-space
 70191 wide-character code is other than a sign or a permissible letter or digit.

70192 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes
 70193 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has
 70194 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for
 70195 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a
 70196 minus-sign, the value resulting from the conversion shall be negated. A pointer to the final
 70197 wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is
 70198 not a null pointer.

- 70199 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
70200 accepted.
- 70201 If the subject sequence is empty or does not have the expected form, no conversion shall be
70202 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
70203 *endptr* is not a null pointer.
- 70204 CX These functions shall not change the setting of *errno* if successful.
- 70205 Since 0, {LONG_MIN} or {LLONG_MIN} and {LONG_MAX} or {LLONG_MAX} are returned on
70206 error and are also valid returns on success, an application wishing to check for error situations
70207 should set *errno* to 0, then call *wcstol()* or *wcstoll()*, then check *errno*.
- 70208 **RETURN VALUE**
- 70209 Upon successful completion, these functions shall return the converted value, if any. If no
70210 CX conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If
70211 the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX},
70212 {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and
70213 *errno* set to [ERANGE].
- 70214 **ERRORS**
- 70215 These functions shall fail if:
- 70216 CX [EINVAL] The value of *base* is not supported.
- 70217 [ERANGE] The value to be returned is not representable.
- 70218 These functions may fail if:
- 70219 CX [EINVAL] No conversion could be performed.
- 70220 **EXAMPLES**
- 70221 None.
- 70222 **APPLICATION USAGE**
- 70223 None.
- 70224 **RATIONALE**
- 70225 None.
- 70226 **FUTURE DIRECTIONS**
- 70227 None.
- 70228 **SEE ALSO**
- 70229 *fscanf()*, *iswalnum()*, *wcstod()*
- 70230 XBD <wchar.h>
- 70231 **CHANGE HISTORY**
- 70232 First released in Issue 4. Derived from the MSE working draft.
- 70233 **Issue 5**
- 70234 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
- 70235 **Issue 6**
- 70236 Extensions beyond the ISO C standard are marked.

wcstol()*System Interfaces*

- 70237 The following new requirements on POSIX implementations derive from alignment with the
70238 Single UNIX Specification:
- 70239 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
70240 added if no conversion could be performed.
- 70241 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
- 70242 • The *wcstol()* prototype is updated.
 - 70243 • The *wcstoll()* function is added.
- 70244 **Issue 7**
- 70245 SD5-XSH-ERN-56 is applied, removing the reference to **unsigned long** and **unsigned long long**
70246 from the DESCRIPTION.

*System Interfaces***wcstold()**70247 **NAME**

70248 wcstold — convert a wide-character string to a double-precision number

70249 **SYNOPSIS**

70250 #include <wchar.h>

70251 long double wcstold(const wchar_t *restrict nptr,

70252 wchar_t **restrict endptr);

70253 **DESCRIPTION**70254 Refer to *wcstod()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

wcstoll()*System Interfaces*70255 **NAME**

70256 wcstoll — convert a wide-character string to a long integer

70257 **SYNOPSIS**

70258 #include <wchar.h>

70259 long long wcstoll(const wchar_t *restrict nptr,

70260 wchar_t **restrict endptr, int base);

70261 **DESCRIPTION**70262 Refer to *wcstol()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

wcstombs()**NAME**

wcstombs — convert a wide-character string to a character string

SYNOPSIS

```
#include <stdlib.h>
```

```
size_t wcstombs(char *restrict s, const wchar_t *restrict pwcs,  
size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *wcstombs()* function shall convert the sequence of wide-character codes that are in the array pointed to by *pwcs* into a sequence of characters that begins in the initial shift state and store these characters into the array pointed to by *s*, stopping if a character would exceed the limit of *n* total bytes or if a null byte is stored. Each wide-character code shall be converted as if by a call to *wctomb()*, except that the shift state of *wctomb()* shall not be affected.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

CX No more than *n* bytes shall be modified in the array pointed to by *s*. If copying takes place between objects that overlap, the behavior is undefined. If *s* is a null pointer, *wcstombs()* shall return the length required to convert the entire array regardless of the value of *n*, but no values are stored.

The *wcstombs()* function need not be thread-safe.

RETURN VALUE

If a wide-character code is encountered that does not correspond to a valid character (of one or more bytes each), *wcstombs()* shall return **(size_t)-1**. Otherwise, *wcstombs()* shall return the number of bytes stored in the character array, not including any terminating null byte. The array shall not be null-terminated if the value returned is *n*.

ERRORS

The *wcstombs()* function shall fail if:

CX **[EILSEQ]** A wide-character code does not correspond to a valid character.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mblen(), *mbtowl()*, *mbstowcs()*, *wctomb()*

XBD *<stdlib.h>*

wcstombs()*System Interfaces*70303 **CHANGE HISTORY**

70304 First released in Issue 4. Derived from the ISO C standard.

70305 **Issue 6**70306 The following new requirements on POSIX implementations derive from alignment with the
70307 Single UNIX Specification:

- 70308 • The DESCRIPTION states the effect of when *s* is a null pointer.
- 70309 • The [EILSEQ] error condition is added.

70310 The *wcstombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.70311 **Issue 7**

70312 Austin Group Interpretations 1003.1-2001 #156 and #170 are applied.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

wcstoul, wcstoull — convert a wide-character string to an unsigned long

SYNOPSIS

```
#include <wchar.h>

unsigned long wcstoul(const wchar_t *restrict nptr,
                     wchar_t **restrict endptr, int base);
unsigned long long wcstoull(const wchar_t *restrict nptr,
                           wchar_t **restrict endptr, int base);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *wcstoul()* and *wcstoull()* functions shall convert the initial portion of the wide-character string pointed to by *nptr* to **unsigned long** and **unsigned long long** representation, respectively. First, they shall decompose the input wide-character string into three parts:

1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by *iswspace()*)
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final wide-character string of one or more unrecognized wide-character codes, including the terminating null wide-character code of the input wide-character string

Then they shall attempt to convert the subject sequence to an unsigned integer, and return the result.

If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* shall be permitted. If the value of *base* is 16, the wide-character codes 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input wide-character string, starting with the first wide-character code that is not white space and is of the expected form. The subject sequence contains no wide-character codes if the input wide-character string is empty or consists entirely of white-space wide-character codes, or if the first wide-character code that is not white space is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus-sign, the value resulting from the conversion shall be negated. A pointer to the final wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is

wcstoul()*System Interfaces*

70359 not a null pointer.

70360 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
70361 accepted.

70362 If the subject sequence is empty or does not have the expected form, no conversion shall be
70363 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
70364 *endptr* is not a null pointer.

70365 CX The *wcstoul()* function shall not change the setting of *errno* if successful.

70366 Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and 0 is also a valid return
70367 on success, an application wishing to check for error situations should set *errno* to 0, then call
70368 *wcstoul()* or *wcstoull()*, then check *errno*.

RETURN VALUE

70370 Upon successful completion, the *wcstoul()* and *wcstoull()* functions shall return the converted
70371 CX value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to
70372 indicate the error. If the correct value is outside the range of representable values,
70373 {ULONG_MAX} or {ULLONG_MAX} respectively shall be returned and *errno* set to [ERANGE].

ERRORS

70374 These functions shall fail if:

70376 CX [EINVAL] The value of *base* is not supported.

70377 [ERANGE] The value to be returned is not representable.

70378 These functions may fail if:

70379 CX [EINVAL] No conversion could be performed.

EXAMPLES

70381 None.

APPLICATION USAGE

70383 None.

RATIONALE

70385 None.

FUTURE DIRECTIONS

70387 None.

SEE ALSO

70389 *fscanf()*, *iswalpha()*, *wcstod()*, *wcstol()*

70390 XBD <wchar.h>

CHANGE HISTORY

70392 First released in Issue 4. Derived from the MSE working draft.

Issue 5

70394 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

70396 Extensions beyond the ISO C standard are marked.

70397 The following new requirements on POSIX implementations derive from alignment with the
70398 Single UNIX Specification:

- 70399 • The [EINVAL] error condition is added for when the value of *base* is not supported.

70400 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
70401 added if no conversion could be performed.

70402 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 70403 • The *wcstoul()* prototype is updated.
- 70404 • The *wcstoull()* function is added.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

wcstoumax()*System Interfaces*70405 **NAME**

70406 wcstoumax — convert a wide-character string to an integer type

70407 **SYNOPSIS**

70408 #include <stddef.h>

70409 #include <inttypes.h>

70410 uintmax_t wcstoumax(const wchar_t *restrict nptr,

70411 wchar_t **restrict endptr, int base);

70412 **DESCRIPTION**70413 Refer to *wcstoimax()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

wcswidth — number of column positions of a wide-character string

SYNOPSIS

XSI `#include <wchar.h>`

```
int wcswidth(const wchar_t *pwcs, size_t n);
```

DESCRIPTION

The *wcswidth()* function shall determine the number of column positions required for *n* wide-character codes (or fewer than *n* wide-character codes if a null wide-character code is encountered before *n* wide-character codes are exhausted) in the string pointed to by *pwcs*.

RETURN VALUE

The *wcswidth()* function either shall return 0 (if *pwcs* points to a null wide-character code), or return the number of column positions to be occupied by the wide-character string pointed to by *pwcs*, or return -1 (if any of the first *n* wide-character codes in the wide-character string pointed to by *pwcs* is not a printable wide-character code).

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the return value for a non-printable wide character is not specified.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[wcwidth\(\)](#)

XBD [Section 3.103](#) (on page 50), [<wchar.h>](#)

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The Open Group Corrigendum U021/11 is applied. The function is marked as an extension.

wcsxfrm()*System Interfaces*70446 **NAME**70447 `wcsxfrm, wcsxfrm_l` — wide-character string transformation70448 **SYNOPSIS**70449 `#include <wchar.h>`70450 `size_t wcsxfrm(wchar_t *restrict ws1, const wchar_t *restrict ws2,`
70451 `size_t n);`70452 CX `size_t wcsxfrm_l(wchar_t *restrict ws1, const wchar_t *restrict ws2,`
70453 `size_t n, locale_t locale);`70454 **DESCRIPTION**70455 CX For `wcsxfrm()`: The functionality described on this reference page is aligned with the ISO C
70456 standard. Any conflict between the requirements described here and the ISO C standard is
70457 unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.70458 CX The `wcsxfrm()` and `wcsxfrm_l()` functions shall transform the wide-character string pointed to
70459 by `ws2` and place the resulting wide-character string into the array pointed to by `ws1`. The
70460 transformation shall be such that if `wcscmp()` is applied to two transformed wide strings, it shall
70461 CX return a value greater than, equal to, or less than 0, corresponding to the result of `wcscoll()` and
70462 `wcscoll_l()` applied to the same two original wide-character strings, and the same `LC_COLLATE`
70463 CX category of the locale of the process or the locale object `locale`, respectively. No more than `n`
70464 wide-character codes shall be placed into the resulting array pointed to by `ws1`, including the
70465 terminating null wide-character code. If `n` is 0, `ws1` is permitted to be a null pointer. If copying
70466 takes place between objects that overlap, the behavior is undefined.70467 CX The `wcsxfrm()` and `wcsxfrm_l()` functions shall not change the setting of `errno` if successful.70468 Since no return value is reserved to indicate an error, an application wishing to check for error
70469 situations should set `errno` to 0, then call `wcsxfrm()` or `wcsxfrm_l()`, then check `errno`.70470 **RETURN VALUE**70471 CX The `wcsxfrm()` and `wcsxfrm_l()` functions shall return the length of the transformed wide-
70472 character string (not including the terminating null wide-character code). If the value returned is
70473 `n` or more, the contents of the array pointed to by `ws1` are unspecified.70474 CX On error, the `wcsxfrm()` and `wcsxfrm_l()` functions may set `errno`, but no return value is reserved
70475 to indicate an error.70476 **ERRORS**

70477 These functions may fail if:

70478 CX [EINVAL] The wide-character string pointed to by `ws2` contains wide-character codes
70479 outside the domain of the collating sequence.70480 The `wcsxfrm_l()` function may fail if:70481 CX [EINVAL] `locale` is not a valid locale object handle.

EXAMPLES

None.

APPLICATION USAGE

The transformation function is such that two transformed wide-character strings can be ordered by *wscmp()* as appropriate to collating sequence information in the locale of the process (category *LC_COLLATE*).

The fact that when *n* is 0 *ws1* is permitted to be a null pointer is useful to determine the size of the *ws1* array prior to making the transformation.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wscmp(), *wscoll()*

XBD <*wchar.h*>

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

In earlier versions, this function was required to return -1 on error.

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The *wcsxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

The *wcsxfrm_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

wctob()

System Interfaces

70513 **NAME**70514 **wctob** — wide-character to single-byte conversion70515 **SYNOPSIS**70516 `#include <stdio.h>`70517 `#include <wchar.h>`70518 `int wctob(wint_t c);`70519 **DESCRIPTION**

70520 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 70521 conflict between the requirements described here and the ISO C standard is unintentional. This
 70522 volume of POSIX.1-2008 defers to the ISO C standard.

70523 The *wctob()* function shall determine whether *c* corresponds to a member of the extended
 70524 character set whose character representation is a single byte when in the initial shift state.

70525 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

70526 **RETURN VALUE**

70527 The *wctob()* function shall return EOF if *c* does not correspond to a character with length one in
 70528 the initial shift state. Otherwise, it shall return the single-byte representation of that character as
 70529 an **unsigned char** converted to **int**.

70530 **ERRORS**

70531 No errors are defined.

70532 **EXAMPLES**

70533 None.

70534 **APPLICATION USAGE**

70535 None.

70536 **RATIONALE**

70537 None.

70538 **FUTURE DIRECTIONS**

70539 None.

70540 **SEE ALSO**70541 *btowc()*70542 XBD *<stdio.h>*, *<wchar.h>*70543 **CHANGE HISTORY**

70544 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 70545 (E).

System Interfaces

wctomb()**NAME**

wctomb — convert a wide-character code to a character

SYNOPSIS

```
#include <stdlib.h>
```

```
int wctomb(char *s, wchar_t wchar);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *wctomb()* function shall determine the number of bytes needed to represent the character corresponding to the wide-character code whose value is *wchar* (including any change in the shift state). It shall store the character representation (possibly multiple bytes and any special bytes to change shift state) in the array object pointed to by *s* (if *s* is not a null pointer). At most {MB_CUR_MAX} bytes shall be stored. If *wchar* is 0, a null byte shall be stored, preceded by any shift sequence needed to restore the initial shift state, and *wctomb()* shall be left in the initial shift state.

The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a state-dependent encoding, this function shall be placed into its initial state by a call for which its character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a null pointer shall cause this function to return a non-zero value if encodings have state dependency, and 0 otherwise. Changing the *LC_CTYPE* category causes the shift state of this function to be unspecified.

The *wctomb()* function need not be thread-safe.

The implementation shall behave as if no function defined in this volume of POSIX.1-2008 calls *wctomb()*.

RETURN VALUE

If *s* is a null pointer, *wctomb()* shall return a non-zero or 0 value, if character encodings, respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *wctomb()* shall return -1 if the value of *wchar* does not correspond to a valid character, or return the number of bytes that constitute the character corresponding to the value of *wchar*.

In no case shall the value returned be greater than the value of the {MB_CUR_MAX} macro.

ERRORS

The *wctomb()* function shall fail if:

CX	[EILSEQ]	An invalid wide-character code is detected.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

wctomb()*System Interfaces*70587 **FUTURE DIRECTIONS**

70588 None.

70589 **SEE ALSO**70590 *mblen()*, *mbtowc()*, *mbstowcs()*, *wcstombs()*70591 XBD **<stdlib.h>**70592 **CHANGE HISTORY**

70593 First released in Issue 4. Derived from the ANSI C standard.

70594 **Issue 6**

70595 Extensions beyond the ISO C standard are marked.

70596 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

70597 **Issue 7**

70598 Austin Group Interpretations 1003.1-2001 #156 and #170 are applied.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

wctrans()70599 **NAME**70600 `wctrans, wctrans_l` — define character mapping70601 **SYNOPSIS**70602 `#include <wctype.h>`70603 `wctrans_t wctrans(const char *charclass);`70604 CX `wctrans_t wctrans_l(const char *charclass, locale_t locale);`70605 **DESCRIPTION**70606 CX For `wctrans()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.70609 CX The `wctrans()` and `wctrans_l()` functions are defined for valid character mapping names identified in the current locale. The `charclass` is a string identifying a generic character mapping name for which codeset-specific information is required. The following character mapping names are defined in all locales: **tolower** and **toupper**.70613 These functions shall return a value of type **wctrans_t**, which can be used as the second argument to subsequent calls of `towctrans()` and `towctrans_l()`.70615 CX The `wctrans()` and `wctrans_l()` functions shall determine values of **wctrans_t** according to the rules of the coded character set defined by character mapping information in the locale of the process or in the locale represented by `locale`, respectively (category `LC_CTYPE`).70618 The values returned by `wctrans()` shall be valid until a call to `setlocale()` that modifies the category `LC_CTYPE`.70620 CX The values returned by `wctrans_l()` shall be valid only in calls to `wctrans_l()` with a locale represented by `locale` with the same `LC_CTYPE` category value.70622 **RETURN VALUE**70623 CX The `wctrans()` and `wctrans_l()` functions shall return 0 and may set `errno` to indicate the error if the given character mapping name is not valid for the current locale (category `LC_CTYPE`); otherwise, they shall return a non-zero object of type **wctrans_t** that can be used in calls to `towctrans()` and `towctrans_l()`.70627 **ERRORS**

70628 These functions may fail if:

70629 CX **[EINVAL]** The character mapping name pointed to by `charclass` is not valid in the current locale.70631 The `wctrans_l()` function may fail if:70632 CX **[EINVAL]** `locale` is not a valid locale object handle.70633 **EXAMPLES**

70634 None.

70635 **APPLICATION USAGE**

70636 None.

70637 **RATIONALE**

70638 None.

wctrans()*System Interfaces*70639 **FUTURE DIRECTIONS**

70640 None.

70641 **SEE ALSO**70642 *towctrans()*70643 XBD *<wctype.h>*70644 **CHANGE HISTORY**

70645 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

70646 **Issue 7**70647 The *wctrans_l()* function is added from The Open Group Technical Standard, 2006, Extended
70648 API Set Part 4.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

wctype()**NAME**

wctype, wctype_l — define character class

SYNOPSIS

```
#include <wctype.h>
```

```
wctype_t wctype(const char *property);
```

```
CX wctype_t wctype_l(const char *property, locale_t locale);
```

DESCRIPTION

CX For *wctype()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

CX The *wctype()* and *wctype_l()* functions are defined for valid character class names as defined in the current locale or in the locale represented by *locale*, respectively.

The *property* argument is a string identifying a generic character class for which codeset-specific type information is required. The following character class names shall be defined in all locales:

alnum	digit	punct
alpha	graph	space
blank	lower	upper
cntrl	print	xdigit

Additional character class names defined in the locale definition file (category *LC_CTYPE*) can also be specified.

These functions shall return a value of type **wctype_t**, which can be used as the second argument to subsequent calls of *iswctype()* and *iswctype_l()*.

CX The *wctype()* and *wctype_l()* functions shall determine values of **wctype_t** according to the rules of the coded character set defined by character type information in the locale of the process or in the locale represented by *locale*, respectively (category *LC_CTYPE*).

The values returned by *wctype()* shall be valid until a call to *setlocale()* that modifies the category *LC_CTYPE*.

CX The values returned by *wctype_l()* shall be valid only in calls to *iswctype_l()* with a locale represented by *locale* with the same *LC_CTYPE* category value.

RETURN VALUE

CX The *wctype()* and *wctype_l()* functions shall return 0 if the given character class name is not valid for the current locale (category *LC_CTYPE*); otherwise, they shall return an object of type **wctype_t** that can be used in calls to *iswctype()* and *iswctype_l()*.

ERRORS

The *wctype_l()* function may fail if:

CX [EINVAL] *locale* is not a valid locale object handle.

wctype()

System Interfaces

70685 **EXAMPLES**

70686 None.

70687 **APPLICATION USAGE**

70688 None.

70689 **RATIONALE**

70690 None.

70691 **FUTURE DIRECTIONS**

70692 None.

70693 **SEE ALSO**70694 [*iswctype\(\)*](#)70695 XBD [**<wctype.h>**](#)70696 **CHANGE HISTORY**

70697 First released in Issue 4.

70698 **Issue 5**

70699 The following change has been made in this version for alignment with
70700 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 70701 • The SYNOPSIS has been changed to indicate that this function and associated data types
70702 are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

70703 **Issue 7**

70704 The *wctype_l()* function is added from The Open Group Technical Standard, 2006, Extended API
70705 Set Part 4.

NAME

wcwidth — number of column positions of a wide-character code

SYNOPSIS

```
#include <wchar.h>

int wcwidth(wchar_t wc);
```

DESCRIPTION

The *wcwidth()* function shall determine the number of column positions required for the wide character *wc*. The application shall ensure that the value of *wc* is a character representable as a **wchar_t**, and is a wide-character code corresponding to a valid character in the current locale.

RETURN VALUE

The *wcwidth()* function shall either return 0 (if *wc* is a null wide-character code), or return the number of column positions to be occupied by the wide-character code *wc*, or return -1 (if *wc* does not correspond to a printable wide-character code).

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the return value for a non-printable wide character is not specified.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcswidth()

XBD **<wchar.h>**

CHANGE HISTORY

First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working draft.

Issue 6

The Open Group Corrigendum U021/12 is applied. This function is marked as an extension.

The normative text is updated to avoid use of the term “must” for application requirements.

wmemchr()

System Interfaces

70739 **NAME**70740 **wmemchr** — find a wide character in memory70741 **SYNOPSIS**70742 `#include <wchar.h>`70743 `wchar_t *wmemchr(const wchar_t *ws, wchar_t wc, size_t n);`70744 **DESCRIPTION**

70745 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 70746 conflict between the requirements described here and the ISO C standard is unintentional. This
 70747 volume of POSIX.1-2008 defers to the ISO C standard.

70748 The *wmemchr()* function shall locate the first occurrence of *wc* in the initial *n* wide characters of
 70749 the object pointed to by *ws*. This function shall not be affected by locale and all **wchar_t** values
 70750 shall be treated identically. The null wide character and **wchar_t** values not corresponding to
 70751 valid characters shall not be treated specially.

70752 If *n* is zero, the application shall ensure that *ws* is a valid pointer and the function behaves as if
 70753 no valid occurrence of *wc* is found.

70754 **RETURN VALUE**

70755 The *wmemchr()* function shall return a pointer to the located wide character, or a null pointer if
 70756 the wide character does not occur in the object.

70757 **ERRORS**

70758 No errors are defined.

70759 **EXAMPLES**

70760 None.

70761 **APPLICATION USAGE**

70762 None.

70763 **RATIONALE**

70764 None.

70765 **FUTURE DIRECTIONS**

70766 None.

70767 **SEE ALSO**70768 *wmemcmp()*, *wmemcpy()*, *wmemmove()*, *wmemset()*70769 XBD *<wchar.h>*70770 **CHANGE HISTORY**

70771 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 70772 (E).

70773 **Issue 6**

70774 The normative text is updated to avoid use of the term “must” for application requirements.

System Interfaces

wmemcmp()**NAME**

wmemcmp — compare wide characters in memory

SYNOPSIS

```
#include <wchar.h>
```

```
int wmemcmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *wmemcmp()* function shall compare the first *n* wide characters of the object pointed to by *ws1* to the first *n* wide characters of the object pointed to by *ws2*. This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function shall behave as if the two objects compare equal.

RETURN VALUE

The *wmemcmp()* function shall return an integer greater than, equal to, or less than zero, respectively, as the object pointed to by *ws1* is greater than, equal to, or less than the object pointed to by *ws2*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr(), *wmemcpy()*, *wmemmove()*, *wmemset()*

XBD *<wchar.h>*

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

wmemcpy()*System Interfaces*70812 **NAME**70813 **wmemcpy** — copy wide characters in memory70814 **SYNOPSIS**70815 `#include <wchar.h>`70816 `wchar_t *wmemcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,`
70817 `size_t n);`70818 **DESCRIPTION**70819 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
70820 conflict between the requirements described here and the ISO C standard is unintentional. This
70821 volume of POSIX.1-2008 defers to the ISO C standard.70822 The *wmemcpy()* function shall copy *n* wide characters from the object pointed to by *ws2* to the
70823 object pointed to by *ws1*. This function shall not be affected by locale and all **wchar_t** values
70824 shall be treated identically. The null wide character and **wchar_t** values not corresponding to
70825 valid characters shall not be treated specially.70826 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function
70827 shall copy zero wide characters.70828 **RETURN VALUE**70829 The *wmemcpy()* function shall return the value of *ws1*.70830 **ERRORS**

70831 No errors are defined.

70832 **EXAMPLES**

70833 None.

70834 **APPLICATION USAGE**

70835 None.

70836 **RATIONALE**

70837 None.

70838 **FUTURE DIRECTIONS**

70839 None.

70840 **SEE ALSO**70841 *wmemchr()*, *wmemcpy()*, *wmemmove()*, *wmemset()*70842 XBD **<wchar.h>**70843 **CHANGE HISTORY**70844 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
70845 (E).70846 **Issue 6**

70847 The normative text is updated to avoid use of the term “must” for application requirements.

70848 The *wmemcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

NAME

wmemmove — copy wide characters in memory with overlapping areas

SYNOPSIS

```
#include <wchar.h>
```

```
wchar_t *wmemmove(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *wmemmove()* function shall copy *n* wide characters from the object pointed to by *ws2* to the object pointed to by *ws1*. Copying shall take place as if the *n* wide characters from the object pointed to by *ws2* are first copied into a temporary array of *n* wide characters that does not overlap the objects pointed to by *ws1* or *ws2*, and then the *n* wide characters from the temporary array are copied into the object pointed to by *ws1*.

This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function shall copy zero wide characters.

RETURN VALUE

The *wmemmove()* function shall return the value of *ws1*.

ERRORS

No errors are defined

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr(), *wmemcmp()*, *wmemcpy()*, *wmemset()*

XBD *<wchar.h>*

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

wmemset()

System Interfaces

70888 **NAME**70889 **wmemset** — set wide characters in memory70890 **SYNOPSIS**70891 `#include <wchar.h>`70892 `wchar_t *wmemset(wchar_t *ws, wchar_t wc, size_t n);`70893 **DESCRIPTION**

70894 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 70895 conflict between the requirements described here and the ISO C standard is unintentional. This
 70896 volume of POSIX.1-2008 defers to the ISO C standard.

70897 The *wmemset()* function shall copy the value of *wc* into each of the first *n* wide characters of the
 70898 object pointed to by *ws*. This function shall not be affected by locale and all **wchar_t** values shall
 70899 be treated identically. The null wide character and **wchar_t** values not corresponding to valid
 70900 characters shall not be treated specially.

70901 If *n* is zero, the application shall ensure that *ws* is a valid pointer, and the function shall copy
 70902 zero wide characters.

70903 **RETURN VALUE**70904 The *wmemset()* functions shall return the value of *ws*.70905 **ERRORS**

70906 No errors are defined.

70907 **EXAMPLES**

70908 None.

70909 **APPLICATION USAGE**

70910 None.

70911 **RATIONALE**

70912 None.

70913 **FUTURE DIRECTIONS**

70914 None.

70915 **SEE ALSO**70916 *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemmove()*70917 XBD **<wchar.h>**70918 **CHANGE HISTORY**

70919 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 70920 (E).

70921 **Issue 6**

70922 The normative text is updated to avoid use of the term “must” for application requirements.

NAME

wordexp, wordfree — perform word expansions

SYNOPSIS

```
#include <wordexp.h>

int wordexp(const char *restrict words, wordexp_t *restrict pwordexp,
            int flags);
void wordfree(wordexp_t *pwordexp);
```

DESCRIPTION

The *wordexp()* function shall perform word expansions as described in XCU Section 2.6 (on page 2305), subject to quoting as described in XCU Section 2.2 (on page 2298), and place the list of expanded words into the structure pointed to by *pwordexp*.

The *words* argument is a pointer to a string containing one or more words to be expanded. The expansions shall be the same as would be performed by the command line interpreter if *words* were the part of a command line representing the arguments to a utility. Therefore, the application shall ensure that *words* does not contain an unquoted <newline> character or any of the unquoted shell special characters ' | ', ' & ', ' ; ', ' < ', ' > ' except in the context of command substitution as specified in XCU Section 2.6.3 (on page 2309). It also shall not contain unquoted parentheses or braces, except in the context of command or variable substitution. The application shall ensure that every member of *words* which it expects to have expanded by *wordexp()* does not contain an unquoted initial comment character. The application shall also ensure that any words which it intends to be ignored (because they begin or continue a comment) are deleted from *words*. If the argument *words* contains an unquoted comment character (<number-sign>) that is the beginning of a token, *wordexp()* shall either treat the comment character as a regular character, or interpret it as a comment indicator and ignore the remainder of *words*.

The structure type **wordexp_t** is defined in the **<wordexp.h>** header and includes at least the following members:

Member Type	Member Name	Description
size_t	<i>we_wordc</i>	Count of words matched by <i>words</i> .
char **	<i>we_wordv</i>	Pointer to list of expanded words.
size_t	<i>we_offs</i>	Slots to reserve at the beginning of <i>pwordexp->we_wordv</i> .

The *wordexp()* function shall store the number of generated words into *pwordexp->we_wordc* and a pointer to a list of pointers to words in *pwordexp->we_wordv*. Each individual field created during field splitting (see XCU Section 2.6.5, on page 2311) or pathname expansion (see XCU Section 2.6.6, on page 2311) shall be a separate word in the *pwordexp->we_wordv* list. The words shall be in order as described in XCU Section 2.6 (on page 2305). The first pointer after the last word pointer shall be a null pointer. The expansion of special parameters described in XCU Section 2.5.2 (on page 2302) is unspecified.

It is the caller's responsibility to allocate the storage pointed to by *pwordexp*. The *wordexp()* function shall allocate other space as needed, including memory pointed to by *pwordexp->we_wordv*. The *wordfree()* function frees any memory associated with *pwordexp* from a previous call to *wordexp()*.

The *flags* argument is used to control the behavior of *wordexp()*. The value of *flags* is the bitwise-inclusive OR of zero or more of the following constants, which are defined in **<wordexp.h>**:

wordexp()*System Interfaces*

70968	WRDE_APPEND	Append words generated to the ones from a previous call to <i>wordexp()</i> .
70969	WRDE_DOOFFS	Make use of <i>pwordexp->we_offs</i> . If this flag is set, <i>pwordexp->we_offs</i> is used to specify how many null pointers to add to the beginning of <i>pwordexp->we_wordv</i> . In other words, <i>pwordexp->we_wordv</i> shall point to <i>pwordexp->we_offs</i> null pointers, followed by <i>pwordexp->we_wordc</i> word pointers, followed by a null pointer.
70970		
70971		
70972		
70973		
70974	WRDE_NOCMD	If the implementation supports the utilities defined in the Shell and Utilities volume of POSIX.1-2008, fail if command substitution, as specified in XCU Section 2.6.3 (on page 2309), is requested.
70975		
70976		
70977	WRDE_REUSE	The <i>pwordexp</i> argument was passed to a previous successful call to <i>wordexp()</i> , and has not been passed to <i>wordfree()</i> . The result shall be the same as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i> without WRDE_REUSE.
70978		
70979		
70980		
70981	WRDE_SHOWERR	Do not redirect <i>stderr</i> to <i>/dev/null</i> .
70982	WRDE_UNDEF	Report error on an attempt to expand an undefined shell variable.
70983		
70984		The WRDE_APPEND flag can be used to append a new set of words to those generated by a previous call to <i>wordexp()</i> . The following rules apply to applications when two or more calls to <i>wordexp()</i> are made with the same value of <i>pwordexp</i> and without intervening calls to <i>wordfree()</i> :
70985		
70986		1. The first such call shall not set WRDE_APPEND. All subsequent calls shall set it.
70987		2. All of the calls shall set WRDE_DOOFFS, or all shall not set it.
70988		3. After the second and each subsequent call, <i>pwordexp->we_wordv</i> shall point to a list containing the following:
70989		
70990		a. Zero or more null pointers, as specified by WRDE_DOOFFS and <i>pwordexp->we_offs</i>
70991		
70992		b. Pointers to the words that were in the <i>pwordexp->we_wordv</i> list before the call, in the same order as before
70993		
70994		c. Pointers to the new words generated by the latest call, in the specified order
70995		4. The count returned in <i>pwordexp->we_wordc</i> shall be the total number of words from all of the calls.
70996		
70997		5. The application can change any of the fields after a call to <i>wordexp()</i> , but if it does it shall reset them to the original value before a subsequent call, using the same <i>pwordexp</i> value, to <i>wordfree()</i> or <i>wordexp()</i> with the WRDE_APPEND or WRDE_REUSE flag.
70998		
70999		
71000		If the implementation supports the utilities defined in the Shell and Utilities volume of POSIX.1-2008, and <i>words</i> contains an unquoted character—<newline>, ' ', ' & ', ' ; ', ' < ', ' > ', ' (', ') ', ' { ', ' } '—in an inappropriate context, <i>wordexp()</i> shall fail, and the number of expanded words shall be 0.
71001		
71002		
71003		
71004		Unless WRDE_SHOWERR is set in <i>flags</i> , <i>wordexp()</i> shall redirect <i>stderr</i> to <i>/dev/null</i> for any utilities executed as a result of command substitution while expanding <i>words</i> . If WRDE_SHOWERR is set, <i>wordexp()</i> may write messages to <i>stderr</i> if syntax errors are detected while expanding <i>words</i> .
71005		
71006		
71007		
71008		The application shall ensure that if WRDE_DOOFFS is set, then <i>pwordexp->we_offs</i> has the same value for each <i>wordexp()</i> call and <i>wordfree()</i> call using a given <i>pwordexp</i> .
71009		

71010 The following constants are defined as error return values:

71011	WRDE_BADCHAR	One of the unquoted characters—<newline>, ' ', ' & ', ' ; ', ' < ', ' > ', ' (', ') ', ' { ', ' } '—appears in <i>words</i> in an inappropriate context.
71012		
71013	WRDE_BADVAL	Reference to undefined shell variable when WRDE_UNDEF is set in <i>flags</i> .
71014	WRDE_CMDSUB	Command substitution requested when WRDE_NOCMD was set in <i>flags</i> .
71015	WRDE_NOSPACE	Attempt to allocate memory failed.
71016	WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated string.
71017		

71018 RETURN VALUE

71019 Upon successful completion, *wordexp()* shall return 0. Otherwise, a non-zero value, as described
 71020 in <**wordexp.h**>, shall be returned to indicate an error. If *wordexp()* returns the value
 71021 WRDE_NOSPACE, then *pwordexp*→*we_wordc* and *pwordexp*→*we_wordp* shall be updated to
 71022 reflect any words that were successfully expanded. In other cases, they shall not be modified.

71023 The *wordfree()* function shall not return a value.

71024 ERRORS

71025 No errors are defined.

71026 EXAMPLES

71027 None.

71028 APPLICATION USAGE

71029 The *wordexp()* function is intended to be used by an application that wants to do all of the shell's
 71030 expansions on a word or words obtained from a user. For example, if the application prompts
 71031 for a filename (or list of filenames) and then uses *wordexp()* to process the input, the user could
 71032 respond with anything that would be valid as input to the shell.

71033 The WRDE_NOCMD flag is provided for applications that, for security or other reasons, want to
 71034 prevent a user from executing shell commands. Disallowing unquoted shell special characters
 71035 also prevents unwanted side effects, such as executing a command or writing a file.

71036 POSIX.1-2008 does not require the *wordexp()* function to be thread-safe if passed an expression
 71037 referencing an environment variable while any other thread is concurrently modifying any
 71038 environment variable; see *exec* (on page 772).

71039 RATIONALE

71040 This function was included as an alternative to *glob()*. There had been continuing controversy
 71041 over exactly what features should be included in *glob()*. It is hoped that by providing *wordexp()*
 71042 (which provides all of the shell word expansions, but which may be slow to execute) and *glob()*
 71043 (which is faster, but which only performs pathname expansion, without tilde or parameter
 71044 expansion) this will satisfy the majority of applications.

71045 While *wordexp()* could be implemented entirely as a library routine, it is expected that most
 71046 implementations run a shell in a subprocess to do the expansion.

71047 Two different approaches have been proposed for how the required information might be
 71048 presented to the shell and the results returned. They are presented here as examples.

71049 One proposal is to extend the *echo* utility by adding a **-q** option. This option would cause *echo* to
 71050 add a <backslash> before each <backslash> and <blank> that occurs within an argument. The
 71051 *wordexp()* function could then invoke the shell as follows:

71052 (void) strcpy(buffer, "echo -q");

wordexp()*System Interfaces*

```

71053     (void) strcat(buffer, words);
71054     if ((flags & WRDE_SHOWERR) == 0)
71055         (void) strcat(buffer, "2>/dev/null");
71056     f = popen(buffer, "r");

```

71057 The *wordexp()* function would read the resulting output, remove unquoted <backslash>
 71058 characters, and break into words at unquoted <blank> characters. If the WRDE_NOCMD flag
 71059 was set, *wordexp()* would have to scan *words* before starting the subshell to make sure that there
 71060 would be no command substitution. In any case, it would have to scan *words* for unquoted
 71061 special characters.

71062 Another proposal is to add the following options to *sh*:

71063 **-w** *wordlist*

71064 This option provides a wordlist expansion service to applications. The words in *wordlist*
 71065 shall be expanded and the following written to standard output:

- 71066 1. The count of the number of words after expansion, in decimal, followed by a null
 71067 byte
- 71068 2. The number of bytes needed to represent the expanded words (not including null
 71069 separators), in decimal, followed by a null byte
- 71070 3. The expanded words, each terminated by a null byte

71071 If an error is encountered during word expansion, *sh* exits with a non-zero status after
 71072 writing the former to report any words successfully expanded

71073 **-P** Run in “protected” mode. If specified with the **-w** option, no command substitution shall
 71074 be performed.

71075 With these options, *wordexp()* could be implemented fairly simply by creating a subprocess
 71076 using *fork()* and executing *sh* using the line:

```

71077 execl(<shell path>, "sh", "-P", "-w", words, (char *)0);

```

71078 after directing standard error to **/dev/null**.

71079 It seemed objectionable for a library routine to write messages to standard error, unless explicitly
 71080 requested, so *wordexp()* is required to redirect standard error to **/dev/null** to ensure that no
 71081 messages are generated, even for commands executed for command substitution. The
 71082 WRDE_SHOWERR flag can be specified to request that error messages be written.

71083 The WRDE_REUSE flag allows the implementation to avoid the expense of freeing and
 71084 reallocating memory, if that is possible. A minimal implementation can call *wordfree()* when
 71085 WRDE_REUSE is set.

71086 FUTURE DIRECTIONS

71087 None.

71088 SEE ALSO

71089 *fnmatch()*, *glob()*

71090 XBD <**wordexp.h**>

71091 XCU Chapter 2 (on page 2297)

71092 **CHANGE HISTORY**

71093 First released in Issue 4. Derived from the ISO POSIX-2 standard.

71094 **Issue 5**

71095 Moved from POSIX2 C-language Binding to BASE.

71096 **Issue 6**

71097 The normative text is updated to avoid use of the term “must” for application requirements.

71098 The **restrict** keyword is added to the *wordexp()* prototype for alignment with the
71099 ISO/IEC 9899:1999 standard.

71100 **Issue 7**

71101 Austin Group Interpretation 1003.1-2001 #148 is applied, adding APPLICATION USAGE.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

wprintf()*System Interfaces*71102 **NAME**71103 **wprintf** — print formatted wide-character output71104 **SYNOPSIS**71105 `#include <stdio.h>`71106 `#include <wchar.h>`71107 `int wprintf(const wchar_t *restrict format, ...);`71108 **DESCRIPTION**71109 Refer to *fwprintf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

System Interfaces

write()71110 **NAME**71111 `pwrite, write` — write on a file71112 **SYNOPSIS**

```
71113 #include <unistd.h>
71114 ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
71115               off_t offset);
71116 ssize_t write(int fildes, const void *buf, size_t nbyte);
```

71117 **DESCRIPTION**

71118 The `write()` function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf* to the
 71119 file associated with the open file descriptor, *fildes*.

71120 Before any action described below is taken, and if *nbyte* is zero and the file is a regular file, the
 71121 `write()` function may detect and return errors as described below. In the absence of errors, or if
 71122 error detection is not performed, the `write()` function shall return zero and have no other results.
 71123 If *nbyte* is zero and the file is not a regular file, the results are unspecified.

71124 On a regular file or other file capable of seeking, the actual writing of data shall proceed from
 71125 the position in the file indicated by the file offset associated with *fildes*. Before successful return
 71126 from `write()`, the file offset shall be incremented by the number of bytes actually written. On a
 71127 regular file, if the position of the last byte written is greater than or equal to the length of the file,
 71128 the length of the file shall be set to this position plus one.

71129 On a file not capable of seeking, writing shall always take place starting at the current position.
 71130 The value of a file offset associated with such a device is undefined.

71131 If the `O_APPEND` flag of the file status flags is set, the file offset shall be set to the end of the file
 71132 prior to each write and no intervening file modification operation shall occur between changing
 71133 the file offset and the write operation.

71134 XSI If a `write()` requests that more bytes be written than there is room for (for example, the file size
 71135 limit of the process or the physical end of a medium), only as many bytes as there is room for
 71136 shall be written. For example, suppose there is space for 20 bytes more in a file before reaching a
 71137 limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes would
 71138 give a failure return (except as noted below).

71139 XSI If the request would cause the file size to exceed the soft file size limit for the process and there
 71140 is no room for any bytes to be written, the request shall fail and the implementation shall
 71141 generate the `SIGXFSZ` signal for the thread.

71142 If `write()` is interrupted by a signal before it writes any data, it shall return `-1` with *errno* set to
 71143 `[EINTR]`.

71144 If `write()` is interrupted by a signal after it successfully writes some data, it shall return the
 71145 number of bytes written.

71146 If the value of *nbyte* is greater than `{SSIZE_MAX}`, the result is implementation-defined.

71147 After a `write()` to a regular file has successfully returned:

- 71148 • Any successful `read()` from each byte position in the file that was modified by that write
 71149 shall return the data specified by the `write()` for that position until such byte positions are
 71150 again modified.
- 71151 • Any subsequent successful `write()` to the same byte position in the file shall overwrite that
 71152 file data.

write()*System Interfaces*

71153		Write requests to a pipe or FIFO shall be handled in the same way as a regular file with the
71154		following exceptions:
71155		• There is no file offset associated with a pipe, hence each write request shall append to the
71156		end of the pipe.
71157		• Write requests of {PIPE_BUF} bytes or less shall not be interleaved with data from other
71158		processes doing writes on the same pipe. Writes of greater than {PIPE_BUF} bytes may
71159		have data interleaved, on arbitrary boundaries, with writes by other processes, whether or
71160		not the O_NONBLOCK flag of the file status flags is set.
71161		• If the O_NONBLOCK flag is clear, a write request may cause the thread to block, but on
71162		normal completion it shall return <i>nbyte</i> .
71163		• If the O_NONBLOCK flag is set, <i>write()</i> requests shall be handled differently, in the
71164		following ways:
71165		— The <i>write()</i> function shall not block the thread.
71166		— A write request for {PIPE_BUF} or fewer bytes shall have the following effect: if there
71167		is sufficient space available in the pipe, <i>write()</i> shall transfer all the data and return
71168		the number of bytes requested. Otherwise, <i>write()</i> shall transfer no data and return
71169		−1 with <i>errno</i> set to [EAGAIN].
71170		— A write request for more than {PIPE_BUF} bytes shall cause one of the following:
71171		— When at least one byte can be written, transfer what it can and return the
71172		number of bytes written. When all data previously written to the pipe is read, it
71173		shall transfer at least {PIPE_BUF} bytes.
71174		— When no data can be written, transfer no data, and return −1 with <i>errno</i> set to
71175		[EAGAIN].
71176		When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-
71177		blocking writes and cannot accept the data immediately:
71178		• If the O_NONBLOCK flag is clear, <i>write()</i> shall block the calling thread until the data can
71179		be accepted.
71180		• If the O_NONBLOCK flag is set, <i>write()</i> shall not block the thread. If some data can be
71181		written without blocking the thread, <i>write()</i> shall write what it can and return the number
71182		of bytes written. Otherwise, it shall return −1 and set <i>errno</i> to [EAGAIN].
71183		Upon successful completion, where <i>nbyte</i> is greater than 0, <i>write()</i> shall mark for update the last
71184		data modification and last file status change timestamps of the file, and if the file is a regular file,
71185		the S_ISUID and S_ISGID bits of the file mode may be cleared.
71186		For regular files, no data transfer shall occur past the offset maximum established in the open
71187		file description associated with <i>filides</i> .
71188		If <i>filides</i> refers to a socket, <i>write()</i> shall be equivalent to <i>send()</i> with no flags set.
71189	SIO	If the O_DSYNC bit has been set, write I/O operations on the file descriptor shall complete as
71190		defined by synchronized I/O data integrity completion.
71191		If the O_SYNC bit has been set, write I/O operations on the file descriptor shall complete as
71192		defined by synchronized I/O file integrity completion.
71193	SHM	If <i>filides</i> refers to a shared memory object, the result of the <i>write()</i> function is unspecified.

71194	TYM	If <i>filides</i> refers to a typed memory object, the result of the <i>write()</i> function is unspecified.
71195	OB XSR	If <i>filides</i> refers to a STREAM, the operation of <i>write()</i> shall be determined by the values of the minimum and maximum <i>nbyte</i> range (packet size) accepted by the STREAM. These values are determined by the topmost STREAM module. If <i>nbyte</i> falls within the packet size range, <i>nbyte</i> bytes shall be written. If <i>nbyte</i> does not fall within the range and the minimum packet size value is 0, <i>write()</i> shall break the buffer into maximum packet size segments prior to sending the data downstream (the last segment may contain less than the maximum packet size). If <i>nbyte</i> does not fall within the range and the minimum value is non-zero, <i>write()</i> shall fail with <i>errno</i> set to [ERANGE]. Writing a zero-length buffer (<i>nbyte</i> is 0) to a STREAMS device sends 0 bytes with 0 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no message and 0 is returned. The process may issue <code>I_SWROPT ioctl()</code> to enable zero-length messages to be sent across the pipe or FIFO. When writing to a STREAM, data messages are created with a priority band of 0. When writing to a STREAM that is not a pipe or FIFO: • If <code>O_NONBLOCK</code> is clear, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), <i>write()</i> shall block until data can be accepted. • If <code>O_NONBLOCK</code> is set and the STREAM cannot accept data, <i>write()</i> shall return -1 and set <i>errno</i> to [EAGAIN]. • If <code>O_NONBLOCK</code> is set and part of the buffer has been written while a condition in which the STREAM cannot accept additional data occurs, <i>write()</i> shall terminate and return the number of bytes written. In addition, <i>write()</i> shall fail if the STREAM head has processed an asynchronous error before the call. In this case, the value of <i>errno</i> does not reflect the result of <i>write()</i>, but reflects the prior error. The <i>pwrite()</i> function shall be equivalent to <i>write()</i>, except that it writes into a given position and does not change the file offset (regardless of whether <code>O_APPEND</code> is set). The first three arguments to <i>pwrite()</i> are the same as <i>write()</i> with the addition of a fourth argument <i>offset</i> for the desired position inside the file.
71196		
71197		
71198		
71199		
71200		
71201		
71202		
71203		
71204		
71205		
71206		RETURN VALUE Upon successful completion, these functions shall return the number of bytes actually written to the file associated with <i>filides</i>. This number shall never be greater than <i>nbyte</i>. Otherwise, -1 shall be returned and <i>errno</i> set to indicate the error. ERRORS These functions shall fail if: [EAGAIN] The <code>O_NONBLOCK</code> flag is set for the file descriptor and the thread would be delayed in the <i>write()</i> operation. [EBADF] The <i>filides</i> argument is not a valid file descriptor open for writing. [EFBIG] An attempt was made to write a file that exceeds the implementation-defined maximum file size or the file size limit of the process, and there was no room for any bytes to be written. [EFBIG] The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>filides</i>.
71207		
71208		
71209		
71210		
71211		
71212		
71213		
71214		
71215		
71216		RETURN VALUE Upon successful completion, these functions shall return the number of bytes actually written to the file associated with <i>filides</i>. This number shall never be greater than <i>nbyte</i>. Otherwise, -1 shall be returned and <i>errno</i> set to indicate the error. ERRORS These functions shall fail if: [EAGAIN] The <code>O_NONBLOCK</code> flag is set for the file descriptor and the thread would be delayed in the <i>write()</i> operation. [EBADF] The <i>filides</i> argument is not a valid file descriptor open for writing. [EFBIG] An attempt was made to write a file that exceeds the implementation-defined maximum file size or the file size limit of the process, and there was no room for any bytes to be written. [EFBIG] The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>filides</i>.
71217		
71218		
71219		
71220		
71221		
71222		
71223		
71224		
71225		
71226		RETURN VALUE Upon successful completion, these functions shall return the number of bytes actually written to the file associated with <i>filides</i>. This number shall never be greater than <i>nbyte</i>. Otherwise, -1 shall be returned and <i>errno</i> set to indicate the error. ERRORS These functions shall fail if: [EAGAIN] The <code>O_NONBLOCK</code> flag is set for the file descriptor and the thread would be delayed in the <i>write()</i> operation. [EBADF] The <i>filides</i> argument is not a valid file descriptor open for writing. [EFBIG] An attempt was made to write a file that exceeds the implementation-defined maximum file size or the file size limit of the process, and there was no room for any bytes to be written. [EFBIG] The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>filides</i>.
71227		
71228		
71229		
71230		
71231		
71232		
71233	XSI	
71234		
71235		
71236		
71237		

write()*System Interfaces*

71238		[EINTR]	The write operation was terminated due to the receipt of a signal, and no data was transferred.
71239			
71240		[EIO]	The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
71241			
71242			
71243			
71244		[ENOSPC]	There was no free space remaining on the device containing the file.
71245		[EPIPE]	An attempt is made to write to a pipe or FIFO that is not open for reading by any process, or that only has one end open. A SIGPIPE signal shall also be sent to the thread.
71246			
71247			
71248	OB XSR	[ERANGE]	The transfer request size was outside the range supported by the STREAMS file associated with <i>fildes</i> .
71249			
71250			The <i>write()</i> function shall fail if:
71251		[EAGAIN] or [EWOULDBLOCK]	
71252			The file descriptor is for a socket, is marked O_NONBLOCK, and write would block.
71253			
71254		[ECONNRESET]	A write was attempted on a socket that is not connected.
71255		[EPIPE]	A write was attempted on a socket that is shut down for writing, or is no longer connected. In the latter case, if the socket is of type SOCK_STREAM, a SIGPIPE signal shall also be sent to the thread.
71256			
71257			
71258			These functions may fail if:
71259	OB XSR	[EINVAL]	The STREAM or multiplexer referenced by <i>fildes</i> is linked (directly or indirectly) downstream from a multiplexer.
71260			
71261		[EIO]	A physical I/O error has occurred.
71262		[ENOBUFFS]	Insufficient resources were available in the system to perform the operation.
71263		[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
71264			
71265	OB XSR	[ENXIO]	A hangup occurred on the STREAM being written to.
71266	OB XSR		A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, <i>errno</i> is set to the value included in the error message.
71267			
71268			The <i>write()</i> function may fail if:
71269		[EACCES]	A write was attempted on a socket and the calling process does not have appropriate privileges.
71270			
71271		[ENETDOWN]	A write was attempted on a socket and the local network interface used to reach the destination is down.
71272			
71273		[ENETUNREACH]	
71274			A write was attempted on a socket and no route to the network is present.
71275			The <i>pwrite()</i> function shall fail and the file pointer remain unchanged if:
71276		[EINVAL]	The <i>offset</i> argument is invalid. The value is negative.

71277 [ESPIPE] *fildes* is associated with a pipe or FIFO.

71278 EXAMPLES

71279 Writing from a Buffer

71280 The following example writes data from the buffer pointed to by *buf* to the file associated with
71281 the file descriptor *fd*.

```
71282 #include <sys/types.h>
71283 #include <string.h>
71284 ...
71285 char buf[20];
71286 size_t nbytes;
71287 ssize_t bytes_written;
71288 int fd;
71289 ...
71290 strcpy(buf, "This is a test\n");
71291 nbytes = strlen(buf);
71292 bytes_written = write(fd, buf, nbytes);
71293 ...
```

71294 APPLICATION USAGE

71295 None.

71296 RATIONALE

71297 See also the RATIONALE section in *read()*.

71298 An attempt to write to a pipe or FIFO has several major characteristics:

- 71299 • *Atomic/non-atomic*: A write is atomic if the whole amount written in one operation is not
71300 interleaved with data from any other process. This is useful when there are multiple
71301 writers sending data to a single reader. Applications need to know how large a write
71302 request can be expected to be performed atomically. This maximum is called {PIPE_BUF}.
71303 This volume of POSIX.1-2008 does not say whether write requests for more than
71304 {PIPE_BUF} bytes are atomic, but requires that writes of {PIPE_BUF} or fewer bytes shall
71305 be atomic.
- 71306 • *Blocking/immediate*: Blocking is only possible with O_NONBLOCK clear. If there is enough
71307 space for all the data requested to be written immediately, the implementation should do
71308 so. Otherwise, the calling thread may block; that is, pause until enough space is available
71309 for writing. The effective size of a pipe or FIFO (the maximum amount that can be written
71310 in one operation without blocking) may vary dynamically, depending on the
71311 implementation, so it is not possible to specify a fixed value for it.

- 71312 • *Complete/partial/deferred*: A write request:

```
71313 int fildes;
71314 size_t nbyte;
71315 ssize_t ret;
71316 char *buf;
71317 ret = write(fildes, buf, nbyte);
71318 may return:
```


write()*System Interfaces*

71319 Complete *ret=nbyte*

71320 Partial *ret<nbyte*

71321 This shall never happen if *nbyte*≤{PIPE_BUF}. If it does happen (with
71322 *nbyte*>{PIPE_BUF}), this volume of POSIX.1-2008 does not guarantee
71323 atomicity, even if *ret*≤{PIPE_BUF}, because atomicity is guaranteed according
71324 to the amount *requested*, not the amount *written*.

71325 Deferred: *ret=-1, errno=[EAGAIN]*

71326 This error indicates that a later request may succeed. It does not indicate that
71327 it *shall* succeed, even if *nbyte*≤{PIPE_BUF}, because if no process reads from
71328 the pipe or FIFO, the write never succeeds. An application could usefully
71329 count the number of times [EAGAIN] is caused by a particular value of
71330 *nbyte*>{PIPE_BUF} and perhaps do later writes with a smaller value, on the
71331 assumption that the effective size of the pipe may have decreased.

71332 Partial and deferred writes are only possible with O_NONBLOCK set.

71333 The relations of these properties are shown in the following tables:

71334 **Write to a Pipe or FIFO with O_NONBLOCK clear**

71335 Immediately Writable:	71335 None	71335 Some	71335 <i>nbyte</i>
71336 <i>nbyte</i> ≤{PIPE_BUF}	71336 Atomic blocking 71337 <i>nbyte</i>	71336 Atomic blocking 71337 <i>nbyte</i>	71336 Atomic immediate 71337 <i>nbyte</i>
71338 <i>nbyte</i> >{PIPE_BUF}	71338 Blocking <i>nbyte</i>	71338 Blocking <i>nbyte</i>	71338 Blocking <i>nbyte</i>

71339 If the O_NONBLOCK flag is clear, a write request shall block if the amount writable
71340 immediately is less than that requested. If the flag is set (by *fcntl()*), a write request shall never
71341 block.

71342 **Write to a Pipe or FIFO with O_NONBLOCK set**

71343 Immediately Writable:	71343 None	71343 Some	71343 <i>nbyte</i>
71344 <i>nbyte</i> ≤{PIPE_BUF}	71344 -1, [EAGAIN]	71344 -1, [EAGAIN]	71344 Atomic <i>nbyte</i>
71345 <i>nbyte</i> >{PIPE_BUF}	71345 -1, [EAGAIN]	71345 < <i>nbyte</i> or -1, 71346 [EAGAIN]	71345 ≤ <i>nbyte</i> or -1, 71346 [EAGAIN]

71347 There is no exception regarding partial writes when O_NONBLOCK is set. With the exception
71348 of writing to an empty pipe, this volume of POSIX.1-2008 does not specify exactly when a partial
71349 write is performed since that would require specifying internal details of the implementation.
71350 Every application should be prepared to handle partial writes when O_NONBLOCK is set and
71351 the requested amount is greater than {PIPE_BUF}, just as every application should be prepared
71352 to handle partial writes on other kinds of file descriptors.

71353 The intent of forcing writing at least one byte if any can be written is to assure that each write
71354 makes progress if there is any room in the pipe. If the pipe is empty, {PIPE_BUF} bytes must be
71355 written; if not, at least some progress must have been made.

71356 Where this volume of POSIX.1-2008 requires -1 to be returned and *errno* set to [EAGAIN], most
71357 historical implementations return zero (with the O_NDELAY flag set, which is the historical
71358 predecessor of O_NONBLOCK, but is not itself in this volume of POSIX.1-2008). The error
71359 indications in this volume of POSIX.1-2008 were chosen so that an application can distinguish
71360 these cases from end-of-file. While *write()* cannot receive an indication of end-of-file, *read()* can,
71361 and the two functions have similar return values. Also, some existing systems (for example,
71362 Eighth Edition) permit a write of zero bytes to mean that the reader should get an end-of-file

indication; for those systems, a return value of zero from *write()* indicates a successful write of an end-of-file indication.

Implementations are allowed, but not required, to perform error checking for *write()* requests of zero bytes.

The concept of a {PIPE_MAX} limit (indicating the maximum number of bytes that can be written to a pipe in a single operation) was considered, but rejected, because this concept would unnecessarily limit application writing.

See also the discussion of O_NONBLOCK in *read()*.

Writes can be serialized with respect to other reads and writes. If a *read()* of file data can be proven (by any means) to occur after a *write()* of the data, it must reflect that *write()*, even if the calls are made by different processes. A similar requirement applies to multiple write operations to the same file position. This is needed to guarantee the propagation of data from *write()* calls to subsequent *read()* calls. This requirement is particularly significant for networked file systems, where some caching schemes violate these semantics.

Note that this is specified in terms of *read()* and *write()*. The XSI extensions *readv()* and *writev()* also obey these semantics. A new “high-performance” write analog that did not follow these serialization requirements would also be permitted by this wording. This volume of POSIX.1-2008 is also silent about any effects of application-level caching (such as that done by *stdio*).

This volume of POSIX.1-2008 does not specify the value of the file offset after an error is returned; there are too many cases. For programming errors, such as [EBADF], the concept is meaningless since no file is involved. For errors that are detected immediately, such as [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however, an updated value would be very useful and is the behavior of many implementations.

This volume of POSIX.1-2008 does not specify behavior of concurrent writes to a file from multiple processes. Applications should use some form of concurrency control.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod(), *creat()*, *dup()*, *fcntl()*, *getrlimit()*, *lseek()*, *open()*, *pipe()*, *read()*, *ulimit()*, *writev()*

XBD [`<limits.h>`](#), [`<stropts.h>`](#), [`<sys/uio.h>`](#), [`<unistd.h>`](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

The *pwrite()* function is added.

Issue 6

The DESCRIPTION states that the *write()* function does not block the thread. Previously this said “process” rather than “thread”.

The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

write()*System Interfaces*

71406	The following new requirements on POSIX implementations derive from alignment with the
71407	Single UNIX Specification:
71408	• The DESCRIPTION now states that if <i>write()</i> is interrupted by a signal after it has
71409	successfully written some data, it returns the number of bytes written. In the POSIX.1-1988
71410	standard, it was optional whether <i>write()</i> returned the number of bytes written, or whether
71411	it returned -1 with <i>errno</i> set to [EINTR]. This is a FIPS requirement.
71412	• The following changes are made to support large files:
71413	— For regular files, no data transfer occurs past the offset maximum established in the
71414	open file description associated with the <i>files</i> .
71415	— A second [EFBIG] error condition is added.
71416	• The [EIO] error condition is added.
71417	• The [EPIPE] error condition is added for when a pipe has only one end open.
71418	• The [ENXIO] optional error condition is added.
71419	Text referring to sockets is added to the DESCRIPTION.
71420	The following changes were made to align with the IEEE P1003.1a draft standard:
71421	• The effect of reading zero bytes is clarified.
71422	The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
71423	<i>write()</i> results are unspecified for typed memory objects.
71424	The following error conditions are added for operations on sockets: [EAGAIN],
71425	[EWOULDBLOCK], [ECONNRESET], [ENOTCONN], and [EPIPE].
71426	The [EIO] error is made optional.
71427	The [ENOBUS] error is added for sockets.
71428	The following error conditions are added for operations on sockets: [EACCES], [ENETDOWN],
71429	and [ENETUNREACH].
71430	The <i>writen()</i> function is split out into a separate reference page.
71431	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/146 is applied, updating text in the
71432	ERRORS section from “a SIGPIPE signal is generated to the calling process” to “a SIGPIPE
71433	signal shall also be sent to the thread”.
71434	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/147 is applied, making a correction to the
71435	RATIONALE.
71436	Issue 7
71437	The <i>pwrite()</i> function is moved from the XSI option to the Base.
71438	Functionality relating to the XSI STREAMS option is marked obsolescent.
71439	SD5-XSH-ERN-160 is applied, updating the DESCRIPTION to clarify the requirements for the
71440	<i>pwrite()</i> function, and to change the use of the phrase “file pointer” to “file offset”.

71441 **NAME**71442 `writev` — write a vector71443 **SYNOPSIS**

```
71444 XSI      #include <sys/uio.h>
71445          ssize_t writev(int fildev, const struct iovec *iov, int iovcnt);
```

71446 **DESCRIPTION**

71447 The `writev()` function shall be equivalent to `write()`, except as described below. The `writev()`
 71448 function shall gather output data from the `iovcnt` buffers specified by the members of the `iov`
 71449 array: `iov[0]`, `iov[1]`, ..., `iov[iovcnt-1]`. The `iovcnt` argument is valid if greater than 0 and less than
 71450 or equal to `{IOV_MAX}`, as defined in `<limits.h>`.

71451 Each `iovec` entry specifies the base address and length of an area in memory from which data
 71452 should be written. The `writev()` function shall always write a complete area before proceeding to
 71453 the next.

71454 If `fildev` refers to a regular file and all of the `iov_len` members in the array pointed to by `iov` are 0,
 71455 `writev()` shall return 0 and have no other effect. For other file types, the behavior is unspecified.

71456 If the sum of the `iov_len` values is greater than `{SSIZE_MAX}`, the operation shall fail and no data
 71457 shall be transferred.

71458 **RETURN VALUE**

71459 Upon successful completion, `writev()` shall return the number of bytes actually written.
 71460 Otherwise, it shall return a value of -1, the file pointer shall remain unchanged, and `errno` shall
 71461 be set to indicate an error.

71462 **ERRORS**71463 Refer to `write()`.71464 In addition, the `writev()` function shall fail if:71465 `[EINVAL]` The sum of the `iov_len` values in the `iov` array would overflow an `ssize_t`.71466 The `writev()` function may fail and set `errno` to:71467 `[EINVAL]` The `iovcnt` argument was less than or equal to 0, or greater than `{IOV_MAX}`.71468 **EXAMPLES**71469 **Writing Data from an Array**

71470 The following example writes data from the buffers specified by members of the `iov` array to the
 71471 file associated with the file descriptor `fd`.

```
71472 #include <sys/types.h>
71473 #include <sys/uio.h>
71474 #include <unistd.h>
71475 ...
71476 ssize_t bytes_written;
71477 int fd;
71478 char *buf0 = "short string\n";
71479 char *buf1 = "This is a longer string\n";
71480 char *buf2 = "This is the longest string in this example\n";
71481 int iocnt;
71482 struct iovec iov[3];
```

writev()

System Interfaces

```
71483     iov[0].iov_base = buf0;
71484     iov[0].iov_len = strlen(buf0);
71485     iov[1].iov_base = buf1;
71486     iov[1].iov_len = strlen(buf1);
71487     iov[2].iov_base = buf2;
71488     iov[2].iov_len = strlen(buf2);
71489     ...
71490     iovcnt = sizeof(iov) / sizeof(struct iovec);
71491     bytes_written = writev(fd, iov, iovcnt);
71492     ...
```

71493 APPLICATION USAGE

71494 None.

71495 RATIONALE

71496 Refer to *write()*.

71497 FUTURE DIRECTIONS

71498 None.

71499 SEE ALSO

71500 *readv()*, *write()*

71501 XBD *<limits.h>*, *<sys/uio.h>*

71502 CHANGE HISTORY

71503 First released in Issue 4, Version 2.

71504 Issue 6

71505 Split out from the *write()* reference page.

*System Interfaces***wscanf()**71506 **NAME**

71507 wscanf — convert formatted wide-character input

71508 **SYNOPSIS**

71509 #include <stdio.h>

71510 #include <wchar.h>

71511 int wscanf(const wchar_t *restrict *format*, ...);71512 **DESCRIPTION**71513 Refer to *fwscanf()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

y0()71514 **NAME**71515 y_0 , y_1 , y_n — Bessel functions of the second kind71516 **SYNOPSIS**

```

71517 XSI      #include <math.h>
71518          double y0(double x);
71519          double y1(double x);
71520          double yn(int n, double x);

```

71521 **DESCRIPTION**

71522 The $y_0()$, $y_1()$, and $y_n()$ functions shall compute Bessel functions of x of the second kind of
 71523 orders 0, 1, and n , respectively.

71524 An application wishing to check for error situations should set *errno* to zero and call
 71525 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 71526 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 71527 zero, an error has occurred.

71528 **RETURN VALUE**

71529 Upon successful completion, these functions shall return the relevant Bessel value of x of the
 71530 second kind.

71531 If x is NaN, NaN shall be returned.

71532 If the x argument to these functions is negative, $-\text{HUGE\_VAL}$ or NaN shall be returned, and a
 71533 domain error may occur.

71534 If x is 0.0, $-\text{HUGE\_VAL}$ shall be returned and a pole error may occur.

71535 If the correct result would cause underflow, 0.0 shall be returned and a range error may occur.

71536 If the correct result would cause overflow, $-\text{HUGE\_VAL}$ or 0.0 shall be returned and a range
 71537 error may occur.

71538 **ERRORS**

71539 These functions may fail if:

71540 Domain Error The value of x is negative.

71541 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 71542 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 71543 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 71544 shall be raised.

71545 Pole Error The value of x is zero.

71546 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 71547 then *errno* shall be set to [ERANGE]. If the integer expression
 71548 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 71549 floating-point exception shall be raised.

71550 Range Error The correct result would cause overflow.

71551 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 71552 then *errno* shall be set to [ERANGE]. If the integer expression
 71553 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 71554 floating-point exception shall be raised.

71555 Range Error The value of x is too large in magnitude, or the correct result would cause
 71556 underflow.

71557 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 71558 then *errno* shall be set to [ERANGE]. If the integer expression
 71559 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 71560 floating-point exception shall be raised.

71561 EXAMPLES

71562 None.

71563 APPLICATION USAGE

71564 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 71565 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

71566 RATIONALE

71567 None.

71568 FUTURE DIRECTIONS

71569 None.

71570 SEE ALSO

71571 *feclearexcept()*, *fetestexcept()*, *isnan()*, *j0()*

71572 XBD Section 4.19 (on page 116), <math.h>

71573 CHANGE HISTORY

71574 First released in Issue 1. Derived from Issue 1 of the SVID.

71575 Issue 5

71576 The DESCRIPTION is updated to indicate how an application should check for an error. This
 71577 text was previously published in the APPLICATION USAGE section.

71578 Issue 6

71579 The normative text is updated to avoid use of the term “must” for application requirements.

71580 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling
 71581 with the ISO/IEC 9899:1999 standard.

71582 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/148 is applied, updating the RETURN
 71583 VALUE and ERRORS sections. The changes are made for consistency with the general rules
 71584 stated in “Treatment of Error Conditions for Mathematical Functions” in the Base Definitions
 71585 volume of POSIX.1-2008.

(blank page)

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

71586

Technical Standard

71587

Vol. 3:

71588

Shell and Utilities, Issue 7

71589

The Open Group

71590

The Institute of Electrical and Electronics Engineers, Inc.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

(blank page)

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Introduction

The Shell and Utilities volume of POSIX.1-2008 describes the commands and utilities offered to application programs by POSIX-conformant systems.

1.1 Relationship to Other Documents

1.1.1 System Interfaces

This subsection describes some of the features provided by the System Interfaces volume of POSIX.1-2008 that are assumed to be globally available on all systems conforming to this volume of POSIX.1-2008. This subsection does not attempt to detail all of the features defined in the System Interfaces volume of POSIX.1-2008 that are required by all of the utilities defined in this volume of POSIX.1-2008; the utility and function descriptions point out additional functionality required to provide the corresponding specific features needed by each.

The following subsections describe frequently used concepts. Many of these concepts are described in the Base Definitions volume of POSIX.1-2008. Utility and function description statements override these defaults when appropriate.

1.1.1.1 Process Attributes

The following process attributes, as described in the System Interfaces volume of POSIX.1-2008, are assumed to be supported for all processes in this volume of POSIX.1-2008:

Controlling Terminal	Real Group ID
Current Working Directory	Real User ID
Effective Group ID	Root Directory
Effective User ID	Saved Set-Group-ID
File Descriptors	Saved Set-User-ID
File Mode Creation Mask	Session Membership
Process Group ID	Supplementary Group IDs
Process ID	

A conforming implementation may include additional process attributes.

1.1.1.2 Concurrent Execution of Processes

The following functionality of the *fork()* function defined in the System Interfaces volume of POSIX.1-2008 shall be available on all systems conforming to this volume of POSIX.1-2008:

1. Independent processes shall be capable of executing independently without either process terminating.

- 71623 2. A process shall be able to create a new process with all of the attributes referenced in
 71624 [Section 1.1.1.1](#) (on page 2279), determined according to the semantics of a call to the *fork()*
 71625 function defined in the System Interfaces volume of POSIX.1-2008 followed by a call in
 71626 the child process to one of the *exec* functions defined in the System Interfaces volume of
 71627 POSIX.1-2008.
- 71628 1.1.1.3 *File Access Permissions*
- 71629 The file access control mechanism described by XBD [Section 4.4](#) (on page 108) shall apply to all
 71630 files on an implementation conforming to this volume of POSIX.1-2008.
- 71631 1.1.1.4 *File Read, Write, and Creation*
- 71632 If a file that does not exist is to be written, it shall be created as described below, unless the
 71633 utility description states otherwise.
- 71634 When a file that does not exist is created, the following features defined in the System Interfaces
 71635 volume of POSIX.1-2008 shall apply unless the utility or function description states otherwise:
- 71636 1. The user ID of the file shall be set to the effective user ID of the calling process.
- 71637 2. The group ID of the file shall be set to the effective group ID of the calling process or the
 71638 group ID of the directory in which the file is being created.
- 71639 3. If the file is a regular file, the permission bits of the file shall be set to:
 71640 $S\_IROTH \mid S\_IWOTH \mid S\_IRGRP \mid S\_IWGRP \mid S\_IRUSR \mid S\_IWUSR$
- 71641 (see the description of *File Modes* in XBD [Chapter 13](#) (on page 219), `<sys/stat.h>`) except
 71642 that the bits specified by the file mode creation mask of the process shall be cleared. If the
 71643 file is a directory, the permission bits shall be set to:
 71644 $S\_IRWXU \mid S\_IRWXG \mid S\_IRWXO$
- 71645 except that the bits specified by the file mode creation mask of the process shall be
 71646 cleared.
- 71647 4. The last data access, last data modification, and last file status change timestamps of the
 71648 file shall be updated as specified in XBD [Section 4.8](#) (on page 109).
- 71649 5. If the file is a directory, it shall be an empty directory; otherwise, the file shall have length
 71650 zero.
- 71651 6. If the file is a symbolic link, the effect shall be undefined unless the {POSIX2_SYMLINKS}
 71652 variable is in effect for the directory in which the symbolic link would be created.
- 71653 7. Unless otherwise specified, the file created shall be a regular file.
- 71654 When an attempt is made to create a file that already exists, the utility shall take the action
 71655 indicated in [Table 1-1](#) (on page 2281) corresponding to the type of the file the utility is trying to
 71656 create and the type of the existing file, unless the utility description states otherwise.

Table 1-1 Actions when Creating a File that Already Exists

Existing Type	New Type											Function Creating New
	B	C	D	F	L	M	P	Q	R	S	T	
A <i>fattach()</i> -ed STREAM	F	F	F	F	F	—	—	—	OF	—	U	N/A
B Block Special	F	F	F	F	F	U	U	U	OF	U	U	<i>mknod()</i> **
C Character Special	F	F	F	F	F	U	U	U	OF	U	U	<i>mknod()</i> **
D Directory	F	F	F	F	F	—	—	—	F	—	U	<i>mkdir()</i>
F FIFO Special File	F	F	F	F	F	—	—	—	O	—	U	<i>mkfifo()</i>
L Symbolic Link	F	F	F	F	F	—	—	—	FL	—	U	<i>symlink()</i>
M Shared Memory	F	F	F	F	F	—	—	—	—	—	U	<i>shm_open()</i>
P Semaphore	F	F	F	F	F	—	—	—	—	—	U	<i>sem_open()</i>
Q Message Queue	F	F	F	F	F	—	—	—	—	—	U	<i>mq_open()</i>
R Regular File	F	F	F	F	F	—	—	—	RF	—	U	<i>open()</i>
S Socket	F	F	F	F	F	—	—	—	—	—	U	<i>bind()</i>
T Typed Memory	F	F	F	F	F	U	U	U	U	U	U	*

The following codes are used in Table 1-1:

- F** Fail. The attempt to create the new file shall fail and the utility shall either continue with its operation or exit immediately with a non-zero exit status, depending on the description of the utility.
- FL** Follow link. Unless otherwise specified, the symbolic link shall be followed as specified for pathname resolution, and the operation performed shall be as if the target of the symbolic link (after all resolution) had been named. If the target of the symbolic link does not exist, it shall be as if that nonexistent target had been named directly.
- O** Open FIFO. When attempting to create a regular file, and the existing file is a FIFO special file:
1. If the FIFO is not already open for reading, the attempt shall block until the FIFO is opened for reading.
 2. Once the FIFO is open for reading, the utility shall open the FIFO for writing and continue with its operation.
- OF** The named file shall be opened with the consequences defined for that file type.
- RF** Regular file. When attempting to create a regular file, and the existing file is a regular file:
1. The user ID, group ID, and permission bits of the file shall not be changed.
 2. The file shall be truncated to zero length.
 3. The last data modification and last file status change timestamps shall be marked for update.
- The effect is implementation-defined unless specified by the utility description.
- U** The effect is unspecified unless specified by the utility description.
- *** There is no portable way to create a file of this type.
- **** Not portable.

When a file is to be appended, the file shall be opened in a manner equivalent to using the `O_APPEND` flag, without the `O_TRUNC` flag, in the *open()* function defined in the System Interfaces volume of POSIX.1-2008.

71699 When a file is to be read or written, the file shall be opened with an access mode corresponding
 71700 to the operation to be performed. If file access permissions deny access, the requested operation
 71701 shall fail.

71702 1.1.1.5 File Removal

71703 When a directory that is the root directory or current working directory of any process is
 71704 removed, the effect is implementation-defined. If file access permissions deny access, the
 71705 requested operation shall fail. Otherwise, when a file is removed:

- 71706 1. Its directory entry shall be removed from the file system.
- 71707 2. The link count of the file shall be decremented.
- 71708 3. If the file is an empty directory (see XBD [Section 3.144](#), on page 56):
 - 71709 a. If no process has the directory open, the space occupied by the directory shall be
 71710 freed and the directory shall no longer be accessible.
 - 71711 b. If one or more processes have the directory open, the directory contents shall be
 71712 preserved until all references to the file have been closed.
- 71713 4. If the file is a directory that is not empty, the last file status change timestamp shall be
 71714 marked for update.
- 71715 5. If the file is not a directory:
 - 71716 a. If the link count becomes zero:
 - 71717 i. If no process has the file open, the space occupied by the file shall be freed
 71718 and the file shall no longer be accessible.
 - 71719 ii. If one or more processes have the file open, the file contents shall be
 71720 preserved until all references to the file have been closed.
 - 71721 b. If the link count is not reduced to zero, the last file status change timestamp shall
 71722 be marked for update.
- 71723 6. The last data modification and last file status change timestamps of the containing
 71724 directory shall be marked for update.

71725 1.1.1.6 File Time Values

71726 All files shall have the three time values described by XBD [Section 4.8](#) (on page 109).

71727 1.1.1.7 File Contents

71728 When a reference is made to the contents of a file, *pathname*, this means the equivalent of all of
 71729 the data placed in the space pointed to by *buf* when performing the *read()* function calls in the
 71730 following operations defined in the System Interfaces volume of POSIX.1-2008:

```
71731 while (read (fildes, buf, nbytes) > 0)
71732     ;
```

71733 If the file is indicated by a *pathname* *pathname*, the file descriptor shall be determined by the
 71734 equivalent of the following operation defined in the System Interfaces volume of POSIX.1-2008:

```
71735 fildes = open (pathname, O_RDONLY);
```

71736 The value of *nbytes* in the above sequence is unspecified; if the file is of a type where the data

*Introduction**Relationship to Other Documents*

- 71737 returned by *read()* would vary with different values, the value shall be one that results in the
 71738 most data being returned.
- 71739 If the *read()* function calls would return an error, it is unspecified whether the contents of the file
 71740 are considered to include any data from offsets in the file beyond where the error would be
 71741 returned.
- 71742 **1.1.1.8 Pathname Resolution**
- 71743 The pathname resolution algorithm, described by XBD [Section 4.12](#) (on page 111), shall be used
 71744 by implementations conforming to this volume of POSIX.1-2008; see also XBD [Section 4.5](#) (on
 71745 page 108).
- 71746 **1.1.1.9 Changing the Current Working Directory**
- 71747 When the current working directory (see XBD [Section 3.122](#), on page 53) is to be changed, unless
 71748 the utility or function description states otherwise, the operation shall succeed unless a call to
 71749 the *chdir()* function defined in the System Interfaces volume of POSIX.1-2008 would fail when
 71750 invoked with the new working directory pathname as its argument.
- 71751 **1.1.1.10 Establish the Locale**
- 71752 The functionality of the *setlocale()* function defined in the System Interfaces volume of
 71753 POSIX.1-2008 shall be available on all systems conforming to this volume of POSIX.1-2008; that
 71754 is, utilities that require the capability of establishing an international operating environment
 71755 shall be permitted to set the specified category of the international environment.
- 71756 **1.1.1.11 Actions Equivalent to Functions**
- 71757 Some utility descriptions specify that a utility performs actions equivalent to a function defined
 71758 in the System Interfaces volume of POSIX.1-2008. Such specifications require only that the
 71759 external effects be equivalent, not that any effect within the utility and visible only to the utility
 71760 be equivalent.
- 71761 **1.1.2 Concepts Derived from the ISO C Standard**
- 71762 Some of the standard utilities perform complex data manipulation using their own procedure
 71763 and arithmetic languages, as defined in their EXTENDED DESCRIPTION or OPERANDS
 71764 sections. Unless otherwise noted, the arithmetic and semantic concepts (precision, type
 71765 conversion, control flow, and so on) shall be equivalent to those defined in the ISO C standard,
 71766 as described in the following sections. Note that there is no requirement that the standard
 71767 utilities be implemented in any particular programming language.
- 71768 **1.1.2.1 Arithmetic Precision and Operations**
- 71769 Integer variables and constants, including the values of operands and option-arguments, used
 71770 by the standard utilities listed in this volume of POSIX.1-2008 shall be implemented as
 71771 equivalent to the ISO C standard **signed long** data type; floating point shall be implemented as
 71772 equivalent to the ISO C standard **double** type. Conversions between types shall be as described
 71773 in the ISO C standard. All variables shall be initialized to zero if they are not otherwise assigned
 71774 by the input to the application.

Arithmetic operators and control flow keywords shall be implemented as equivalent to those in the cited ISO C standard section, as listed in [Table 1-2](#).

Table 1-2 Selected ISO C Standard Operators and Control Flow Keywords

Operation	ISO C Standard Equivalent Reference
()	Section 6.5.1, Primary Expressions
postfix ++ postfix --	Section 6.5.2, Postfix Operators
unary + unary - prefix ++ prefix -- ~ ! sizeof()	Section 6.5.3, Unary Operators
* / %	Section 6.5.5, Multiplicative Operators
+ -	Section 6.5.6, Additive Operators
<< >>	Section 6.5.7, Bitwise Shift Operators
<, <= >, >=	Section 6.5.8, Relational Operators
== !=	Section 6.5.9, Equality Operators
&	Section 6.5.10, Bitwise AND Operator
^	Section 6.5.11, Bitwise Exclusive OR Operator
	Section 6.5.12, Bitwise Inclusive OR Operator
&&	Section 6.5.13, Logical AND Operator
	Section 6.5.14, Logical OR Operator
expr?expr:expr	Section 6.5.15, Conditional Operator
=, *=, /=, %= <<=, >>=, &=, ^=, =	Section 6.5.16, Assignment Operators
if () if () ... else switch ()	Section 6.8.4, Selection Statements
while () do ... while () for ()	Section 6.8.5, Iteration Statements
goto continue break return	Section 6.8.6, Jump Statements

Introduction

Relationship to Other Documents

71818 The evaluation of arithmetic expressions shall be equivalent to that described in Section 6.5,
 71819 Expressions, of the ISO C standard.

71820 1.1.2.2 Mathematical Functions

71821 Any mathematical functions with the same names as those in the following sections of the ISO C
 71822 standard:

- 71823 • Section 7.12, Mathematics, <math.h>
- 71824 • Section 7.20.2, Pseudo-Random Sequence Generation Functions

71825 shall be implemented to return the results equivalent to those returned from a call to the
 71826 corresponding function described in the ISO C standard.

71827 1.2 Utility Limits

71828 This section lists magnitude limitations imposed by a specific implementation. The braces
 71829 notation, {LIMIT}, is used in this volume of POSIX.1-2008 to indicate these values, but the braces
 71830 are not part of the name.

71831 **Table 1-3** Utility Limit Minimum Values

Name	Description	Value
{POSIX2_BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	2 048
{POSIX2_BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	1 000
{POSIX2_COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> order keyword in the locale definition file; see the border_start keyword in XBD Section 7.3.2 (on page 146).	2
{POSIX2_EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	32
{POSIX2_LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	2 048
{POSIX2_RE_DUP_MAX}	The maximum number of repeated occurrences of a BRE permitted when using the interval notation $\{m,n\}$; see XBD Section 9.3.6 (on page 186).	255

71857 The values specified in Table 1-3 represent the lowest values conforming implementations shall
 71858 provide and, consequently, the largest values on which an application can rely without further

enquiries, as described below. These values shall be accessible to applications via the *getconf* utility (see *getconf*, on page 2772).

Implementations may provide more liberal, or less restrictive, values than shown in Table 1-3 (on page 2285). These possibly more liberal values are accessible using the symbols in Table 1-4.

The *sysconf()* function defined in the System Interfaces volume of POSIX.1-2008 or the *getconf* utility return the value of each symbol on each specific implementation. The value so retrieved is the largest, or most liberal, value that is available throughout the session lifetime, as determined at session creation. The literal names shown in the table apply only to the *getconf* utility; the high-level language binding describes the exact form of each name to be used by the interfaces in that binding.

All numeric limits defined by the System Interfaces volume of POSIX.1-2008, such as {PATH_MAX}, shall also apply to this volume of POSIX.1-2008. All the utilities defined by this volume of POSIX.1-2008 are implicitly limited by these values, unless otherwise noted in the utility descriptions.

It is not guaranteed that the application can actually reach the specified limit of an implementation in any given case, or at all, as a lack of virtual memory or other resources may prevent this. The limit value indicates only that the implementation does not specifically impose any arbitrary, more restrictive limit.

Table 1-4 Symbolic Utility Limits

Name	Description	Minimum Value
{BC_BASE_MAX}	The maximum <i>base</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_BASE_MAX}
{BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	{POSIX2_BC_DIM_MAX}
{BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_SCALE_MAX}
{BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	{POSIX2_BC_STRING_MAX}
{COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> order keyword in the locale definition file; see the order_start keyword in XBD Section 7.3.2 (on page 146).	{POSIX2_COLL_WEIGHTS_MAX}
{EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	{POSIX2_EXPR_NEST_MAX}

Name	Description	Minimum Value
{LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	{POSIX2_LINE_MAX}
{RE_DUP_MAX}	The maximum number of repeated occurrences of a BRE permitted when using the interval notation $\{m,n\}$; see XBD Section 9.3.6 (on page 186).	{POSIX2_RE_DUP_MAX}

The following value may be a constant within an implementation or may vary from one pathname to another.

{POSIX2_SYMLINKS}

When referring to a directory, the system supports the creation of symbolic links within that directory; for non-directory files, the meaning of {POSIX2_SYMLINKS} is undefined.

1.3 Grammar Conventions

Portions of this volume of POSIX.1-2008 are expressed in terms of a special grammar notation. It is used to portray the complex syntax of certain program input. The grammar is based on the syntax used by the *yacc* utility. However, it does not represent fully functional *yacc* input, suitable for program use; the lexical processing and all semantic requirements are described only in textual form. The grammar is not based on source used in any traditional implementation and has not been tested with the semantic code that would normally be required to accompany it. Furthermore, there is no implication that the partial *yacc* code presented represents the most efficient, or only, means of supporting the complex syntax within the utility. Implementations may use other programming languages or algorithms, as long as the syntax supported is the same as that represented by the grammar.

The following typographical conventions are used in the grammar; they have no significance except to aid in reading.

- The identifiers for the reserved words of the language are shown with a leading capital letter. (These are terminals in the grammar; for example, **While**, **Case**.)
- The identifiers for terminals in the grammar are all named with uppercase letters and underscores; for example, **NEWLINE**, **ASSIGN_OP**, **NAME**.
- The identifiers for non-terminals are all lowercase.

1.4 Utility Description Defaults

This section describes all of the subsections used within the utility descriptions, including:

- Intended usage of the section
- Global defaults that affect all the standard utilities
- The meanings of notations used in this volume of POSIX.1-2008 that are specific to individual utility sections

NAME

This section gives the name or names of the utility and briefly states its purpose.

SYNOPSIS

The SYNOPSIS section summarizes the syntax of the calling sequence for the utility, including options, option-arguments, and operands. Standards for utility naming are described in XBD [Section 12.2](#) (on page 215); for describing the utility's arguments in XBD [Section 12.1](#) (on page 213).

DESCRIPTION

The DESCRIPTION section describes the actions of the utility. If the utility has a very complex set of subcommands or its own procedural language, an EXTENDED DESCRIPTION section is also provided. Most explanations of optional functionality are omitted here, as they are usually explained in the OPTIONS section.

As stated in [Section 1.1.1.11](#) (on page 2283), some functions are described in terms of equivalent functionality. When specific functions are cited, the implementation shall provide equivalent functionality including side-effects associated with successful execution of the function. The treatment of errors and intermediate results from the individual functions cited is generally not specified by this volume of POSIX.1-2008. See the utility's EXIT STATUS and CONSEQUENCES OF ERRORS sections for all actions associated with errors encountered by the utility.

OPTIONS

The OPTIONS section describes the utility options and option-arguments, and how they modify the actions of the utility. Standard utilities that have options either fully comply with XBD [Section 12.2](#) (on page 215) or describe all deviations. Apparent disagreements between functionality descriptions in the OPTIONS and DESCRIPTION (or EXTENDED DESCRIPTION) sections are always resolved in favor of the OPTIONS section.

Each OPTIONS section that uses the phrase "The ... utility shall conform to the Utility Syntax Guidelines ..." refers only to the use of the utility as specified by this volume of POSIX.1-2008; implementation extensions should also conform to the guidelines, but may allow exceptions for historical practice.

Unless otherwise stated in the utility description, when given an option unrecognized by the implementation, or when a required option-argument is not provided, standard utilities shall issue a diagnostic message to standard error and exit with a non-zero exit status.

All utilities in this volume of POSIX.1-2008 shall be capable of processing arguments using eight-bit transparency.

Default Behavior: When this section is listed as "None.", it means that the implementation need not support any options. Standard utilities that do not accept options, but that do accept operands, shall recognize "--" as a first argument to be discarded.

Introduction

Utility Description Defaults

The requirement for recognizing "--" is because conforming applications need a way to shield their operands from any arbitrary options that the implementation may provide as an extension. For example, if the standard utility *foo* is listed as taking no options, and the application needed to give it a pathname with a leading <hyphen>, it could safely do it as:

```
foo -- -myfile
```

and avoid any problems with *-m* used as an extension.

OPERANDS

The OPERANDS section describes the utility operands, and how they affect the actions of the utility. Apparent disagreements between functionality descriptions in the OPERANDS and DESCRIPTION (or EXTENDED DESCRIPTION) sections shall be resolved in favor of the OPERANDS section.

If an operand naming a file can be specified as *'-'*, which means to use the standard input instead of a named file, this is explicitly stated in this section. Unless otherwise stated, the use of multiple instances of *'-'* to mean standard input in a single command produces unspecified results.

Unless otherwise stated, the standard utilities that accept operands shall process those operands in the order specified in the command line.

Default Behavior: When this section is listed as "None.", it means that the implementation need not support any operands.

STDIN

The STDIN section describes the standard input of the utility. This section is frequently merely a reference to the following section, as many utilities treat standard input and input files in the same manner. Unless otherwise stated, all restrictions described in the INPUT FILES section shall apply to this section as well.

Use of a terminal for standard input can cause any of the standard utilities that read standard input to stop when used in the background. For this reason, applications should not use interactive features in scripts to be placed in the background.

The specified standard input format of the standard utilities shall not depend on the existence or value of the environment variables defined in this volume of POSIX.1-2008, except as provided by this volume of POSIX.1-2008.

Default Behavior: When this section is listed as "Not used.", it means that the standard input shall not be read when the utility is used as described by this volume of POSIX.1-2008.

INPUT FILES

The INPUT FILES section describes the files, other than the standard input, used as input by the utility. It includes files named as operands and option-arguments as well as other files that are referred to, such as start-up and initialization files, databases, and so on. Commonly-used files are generally described in one place and cross-referenced by other utilities.

All utilities in this volume of POSIX.1-2008 shall be capable of processing input files using eight-bit transparency.

When a standard utility reads a seekable input file and terminates without an error before it reaches end-of-file, the utility shall ensure that the file offset in the open file description is properly positioned just past the last byte processed by the utility. For files that are not seekable, the state of the file offset in the open file description for that

file is unspecified. A conforming application shall not assume that the following three commands are equivalent:

```
tail -n +2 file
(sed -n 1q; cat) < file
cat file | (sed -n 1q; cat)
```

The second command is equivalent to the first only when the file is seekable. The third command leaves the file offset in the open file description in an unspecified state. Other utilities, such as *head*, *read*, and *sh*, have similar properties.

Some of the standard utilities, such as filters, process input files a line or a block at a time and have no restrictions on the maximum input file size. Some utilities may have size limitations that are not as obvious as file space or memory limitations. Such limitations should reflect resource limitations of some sort, not arbitrary limits set by implementors. Implementations shall document those utilities that are limited by constraints other than file system space, available memory, and other limits specifically cited by this volume of POSIX.1-2008, and identify what the constraint is and indicate a way of estimating when the constraint would be reached. Similarly, some utilities descend the directory tree (recursively). Implementations shall also document any limits that they may have in descending the directory tree that are beyond limits cited by this volume of POSIX.1-2008.

When an input file is described as a “text file”, the utility produces undefined results if given input that is not from a text file, unless otherwise stated. Some utilities (for example, *make*, *read*, *sh*) allow for continued input lines using an escaped <newline> convention; unless otherwise stated, the utility need not be able to accumulate more than {LINE_MAX} bytes from a set of multiple, continued input lines. Thus, for a conforming application the total of all the continued lines in a set cannot exceed {LINE_MAX}. If a utility using the escaped <newline> convention detects an end-of-file condition immediately after an escaped <newline>, the results are unspecified.

Record formats are described in a notation similar to that used by the C-language function, *printf()*. See XBD Chapter 5 (on page 121) for a description of this notation. The format description is intended to be sufficiently rigorous to allow other applications to generate these input files. However, since <blank>s can legitimately be included in some of the fields described by the standard utilities, particularly in locales other than the POSIX locale, this intent is not always realized.

Default Behavior: When this section is listed as “None.”, it means that no input files are required to be supplied when the utility is used as described by this volume of POSIX.1-2008.

ENVIRONMENT VARIABLES

The ENVIRONMENT VARIABLES section lists what variables affect the utility’s execution.

The entire manner in which environment variables described in this volume of POSIX.1-2008 affect the behavior of each utility is described in the ENVIRONMENT VARIABLES section for that utility, in conjunction with the global effects of the *LANG*, *LC_ALL*, and *NLSPATH* environment variables described in XBD Chapter 8 (on page 173). The existence or value of environment variables described in this volume of POSIX.1-2008 shall not otherwise affect the specified behavior of the standard utilities. Any effects of the existence or value of environment variables not described by this volume of POSIX.1-2008 upon the standard utilities are unspecified.

For those standard utilities that use environment variables as a means for selecting a

Introduction

Utility Description Defaults

72080 utility to execute (such as *CC* in *make*), the string provided to the utility is subjected to
 72081 the path search described for *PATH* in XBD [Chapter 8](#) (on page 173).

72082 All utilities in this volume of POSIX.1-2008 shall be capable of processing environment
 72083 variable names and values using eight-bit transparency.

72084 **Default Behavior:** When this section is listed as “None.”, it means that the behavior of
 72085 the utility is not directly affected by environment variables described by this volume of
 72086 POSIX.1-2008 when the utility is used as described by this volume of POSIX.1-2008.

72087 **ASYNCHRONOUS EVENTS**

72088 The ASYNCHRONOUS EVENTS section lists how the utility reacts to such events as
 72089 signals and what signals are caught.

72090 **Default Behavior:** When this section is listed as “Default.”, or it refers to “the standard
 72091 action for all other signals; see [Section 1.4](#) (on page 2288)” it means that the action taken
 72092 as a result of the signal shall be one of the following:

- 72093 1. The action shall be that inherited from the parent according to the rules of
 72094 inheritance of signal actions defined in the System Interfaces volume of
 72095 POSIX.1-2008.
- 72096 2. When no action has been taken to change the default, the default action shall be
 72097 that specified by the System Interfaces volume of POSIX.1-2008.
- 72098 3. The result of the utility’s execution is as if default actions had been taken.

72099 A utility is permitted to catch a signal, perform some additional processing (such as
 72100 deleting temporary files), restore the default signal action (or action inherited from the
 72101 parent process), and resignal itself.

72102 **STDOUT**

72103 The STDOUT section completely describes the standard output of the utility. This
 72104 section is frequently merely a reference to the following section, OUTPUT FILES,
 72105 because many utilities treat standard output and output files in the same manner.

72106 Use of a terminal for standard output may cause any of the standard utilities that write
 72107 standard output to stop when used in the background. For this reason, applications
 72108 should not use interactive features in scripts to be placed in the background.

72109 Record formats are described in a notation similar to that used by the C-language
 72110 function, *printf()*. See XBD [Chapter 5](#) (on page 121) for a description of this notation.

72111 The specified standard output of the standard utilities shall not depend on the
 72112 existence or value of the environment variables defined in this volume of POSIX.1-2008,
 72113 except as provided by this volume of POSIX.1-2008.

72114 Some of the standard utilities describe their output using the verb *display*, defined in
 72115 XBD [Section 3.133](#) (on page 54). Output described in the STDOUT sections of such
 72116 utilities may be produced using means other than standard output. When standard
 72117 output is directed to a terminal, the output described shall be written directly to the
 72118 terminal. Otherwise, the results are undefined.

72119 **Default Behavior:** When this section is listed as “Not used.”, it means that the standard
 72120 output shall not be written when the utility is used as described by this volume of
 72121 POSIX.1-2008.

72122 **STDERR**

72123 The STDERR section describes the standard error output of the utility. Only those
 72124 messages that are purposely sent by the utility are described.

72125 Use of a terminal for standard error may cause any of the standard utilities that write
 72126 standard error output to stop when used in the background. For this reason,
 72127 applications should not use interactive features in scripts to be placed in the
 72128 background.

72129 The format of diagnostic messages for most utilities is unspecified, but the language
 72130 and cultural conventions of diagnostic and informative messages whose format is
 72131 unspecified by this volume of POSIX.1-2008 should be affected by the setting of
 72132 XSI *LC_MESSAGES* and *NLSPATH*.

72133 The specified standard error output of standard utilities shall not depend on the
 72134 existence or value of the environment variables defined in this volume of POSIX.1-2008,
 72135 except as provided by this volume of POSIX.1-2008.

72136 **Default Behavior:** When this section is listed as “The standard error shall be used only
 72137 for diagnostic messages.”, it means that, unless otherwise stated, the diagnostic
 72138 messages shall be sent to the standard error only when the exit status indicates that an
 72139 error occurred and the utility is used as described by this volume of POSIX.1-2008.

72140 When this section is listed as “Not used.”, it means that the standard error shall not be
 72141 used when the utility is used as described in this volume of POSIX.1-2008.

72142 OUTPUT FILES

72143 The OUTPUT FILES section completely describes the files created or modified by the
 72144 utility. Temporary or system files that are created for internal usage by this utility or
 72145 other parts of the implementation (for example, spool, log, and audit files) are not
 72146 described in this, or any, section. The utilities creating such files and the names of such
 72147 files are unspecified. If applications are written to use temporary or intermediate files,
 72148 they should use the *TMPDIR* environment variable, if it is set and represents an
 72149 accessible directory, to select the location of temporary files.

72150 Implementations shall ensure that temporary files, when used by the standard utilities,
 72151 are named so that different utilities or multiple instances of the same utility can operate
 72152 simultaneously without regard to their working directories, or any other process
 72153 characteristic other than process ID. There are two exceptions to this rule:

- 72154 1. Resources for temporary files other than the name space (for example, disk
 72155 space, available directory entries, or number of processes allowed) are not
 72156 guaranteed.
- 72157 2. Certain standard utilities generate output files that are intended as input for
 72158 other utilities (for example, *lex* generates *lex.yy.c*), and these cannot have unique
 72159 names. These cases are explicitly identified in the descriptions of the respective
 72160 utilities.

72161 Any temporary file created by the implementation shall be removed by the
 72162 implementation upon a utility's successful exit, exit because of errors, or before
 72163 termination by any of the *SIGHUP*, *SIGINT*, or *SIGTERM* signals, unless specified
 72164 otherwise by the utility description.

72165 Receipt of the *SIGQUIT* signal should generally cause termination (unless in some
 72166 debugging mode) that would bypass any attempted recovery actions.

72167 Record formats are described in a notation similar to that used by the C-language
 72168 function, *printf()*; see XBD Chapter 5 (on page 121) for a description of this notation.

72169 **Default Behavior:** When this section is listed as “None.”, it means that no files are
 72170 created or modified as a consequence of direct action on the part of the utility when the
 72171 utility is used as described by this volume of POSIX.1-2008. However, the utility may

72172 create or modify system files, such as log files, that are outside the utility's normal
72173 execution environment.

72174 EXTENDED DESCRIPTION

72175 The EXTENDED DESCRIPTION section provides a place for describing the actions of
72176 very complicated utilities, such as text editors or language processors, which typically
72177 have elaborate command languages.

72178 **Default Behavior:** When this section is listed as "None.", no further description is
72179 necessary.

72180 EXIT STATUS

72181 The EXIT STATUS section describes the values the utility shall return to the calling
72182 program, or shell, and the conditions that cause these values to be returned. Usually,
72183 utilities return zero for successful completion and values greater than zero for various
72184 error conditions. If specific numeric values are listed in this section, the system shall
72185 use those values for the errors described. In some cases, status values are listed more
72186 loosely, such as >0. A strictly conforming application shall not rely on any specific
72187 value in the range shown and shall be prepared to receive any value in the range.

72188 For example, a utility may list zero as a successful return, 1 as a failure for a specific
72189 reason, and >1 as "an error occurred". In this case, unspecified conditions may cause a
72190 2 or 3, or other value, to be returned. A conforming application should be written so
72191 that it tests for successful exit status values (zero in this case), rather than relying upon
72192 the single specific error value listed in this volume of POSIX.1-2008. In that way, it has
72193 maximum portability, even on implementations with extensions.

72194 Unspecified error conditions may be represented by specific values not listed in this
72195 volume of POSIX.1-2008.

72196 CONSEQUENCES OF ERRORS

72197 The CONSEQUENCES OF ERRORS section describes the effects on the environment,
72198 file systems, process state, and so on, when error conditions occur. It does not describe
72199 error messages produced or exit status values used.

72200 The many reasons for failure of a utility are generally not specified by the utility
72201 descriptions. Utilities may terminate prematurely if they encounter: invalid usage of
72202 options, arguments, or environment variables; invalid usage of the complex syntaxes
72203 expressed in EXTENDED DESCRIPTION sections; difficulties accessing, creating,
72204 reading, or writing files; or difficulties associated with the privileges of the process.

72205 The following shall apply to each utility, unless otherwise stated:

- 72206 • If the requested action cannot be performed on an operand representing a file,
72207 directory, user, process, and so on, the utility shall issue a diagnostic message to
72208 standard error and continue processing the next operand in sequence, but the
72209 final exit status shall be returned as non-zero.

72210 For a utility that recursively traverses a file hierarchy (such as *find* or *chown -R*), if
72211 the requested action cannot be performed on a file or directory encountered in the
72212 hierarchy, the utility shall issue a diagnostic message to standard error and
72213 continue processing the remaining files in the hierarchy, but the final exit status
72214 shall be returned as non-zero.

- 72215 • If the requested action characterized by an option or option-argument cannot be
72216 performed, the utility shall issue a diagnostic message to standard error and the
72217 exit status returned shall be non-zero.

72218 • When an unrecoverable error condition is encountered, the utility shall exit with a
72219 non-zero exit status.

72220 • A diagnostic message shall be written to standard error whenever an error
72221 condition occurs.

72222 When a utility encounters an error condition several actions are possible, depending on
72223 the severity of the error and the state of the utility. Included in the possible actions of
72224 various utilities are: deletion of temporary or intermediate work files; deletion of
72225 incomplete files; validity checking of the file system or directory.

72226 **Default Behavior:** When this section is listed as “Default.”, it means that any changes
72227 to the environment are unspecified.

72228 APPLICATION USAGE

72229 This section is informative.

72230 The APPLICATION USAGE section gives advice to the application programmer or
72231 user about the way the utility should be used.

72232 EXAMPLES

72233 This section is informative.

72234 The EXAMPLES section gives one or more examples of usage, where appropriate. In
72235 the event of conflict between an example and a normative part of the specification, the
72236 normative material is to be taken as correct.

72237 In all examples, quoting has been used, showing how sample commands (utility names
72238 combined with arguments) could be passed correctly to a shell (see *sh*) or as a string to
72239 the *system()* function defined in the System Interfaces volume of POSIX.1-2008. Such
72240 quoting would not be used if the utility is invoked using one of the *exec* functions
72241 defined in the System Interfaces volume of POSIX.1-2008.

72242 RATIONALE

72243 This section is informative.

72244 This section contains historical information concerning the contents of this volume of
72245 POSIX.1-2008 and why features were included or discarded by the standard
72246 developers.

72247 FUTURE DIRECTIONS

72248 This section is informative.

72249 The FUTURE DIRECTIONS section should be used as a guide to current thinking; there
72250 is not necessarily a commitment to implement all of these future directions in their
72251 entirety.

72252 SEE ALSO

72253 This section is informative.

72254 The SEE ALSO section lists related entries.

72255 CHANGE HISTORY

72256 This section is informative.

72257 This section shows the derivation of the entry and any significant changes that have
72258 been made to it.

72259 Certain of the standard utilities describe how they can invoke other utilities or applications, such
72260 as by passing a command string to the command interpreter. The external influences (STDIN,
72261 ENVIRONMENT VARIABLES, and so on) and external effects (STDOUT, CONSEQUENCES OF

Introduction

Utility Description Defaults

72262 ERRORS, and so on) of such invoked utilities are not described in the section concerning the
 72263 standard utility that invokes them.

72264 1.5 Considerations for Utilities in Support of Files of Arbitrary Size

72265 The following utilities support files of any size up to the maximum that can be created by the
 72266 implementation. This support includes correct writing of file size-related values (such as file
 72267 sizes and offsets, line numbers, and block counts) and correct interpretation of command line
 72268 arguments that contain such values.

72269	<i>basename</i>	Return non-directory portion of pathname.
72270	<i>cat</i>	Concatenate and print files.
72271	<i>cd</i>	Change working directory.
72272	<i>chgrp</i>	Change file group ownership.
72273	<i>chmod</i>	Change file modes.
72274	<i>chown</i>	Change file ownership.
72275	<i>cksum</i>	Write file checksums and sizes.
72276	<i>cmp</i>	Compare two files.
72277	<i>cp</i>	Copy files.
72278	<i>dd</i>	Convert and copy a file.
72279	<i>df</i>	Report free disk space.
72280	<i>dirname</i>	Return directory portion of pathname.
72281	<i>du</i>	Estimate file space usage.
72282	<i>find</i>	Find files.
72283	<i>ln</i>	Link files.
72284	<i>ls</i>	List directory contents.
72285	<i>mkdir</i>	Make directories.
72286	<i>mv</i>	Move files.
72287	<i>pathchk</i>	Check pathnames.
72288	<i>pwd</i>	Return working directory name.
72289	<i>rm</i>	Remove directory entries.
72290	<i>rmdir</i>	Remove directories.
72291	<i>sh</i>	Shell, the standard command language interpreter.
72292	<i>sum</i>	Print checksum and block or byte count of a file.
72293	<i>test</i>	Evaluate expression.
72294	<i>touch</i>	Change file access and modification times.
72295	<i>ulimit</i>	Set or report file size limit.

- 72296
- 72297
- 72298
- 72299
- 72300
- 72301
- Exceptions to the requirement that utilities support files of any size up to the maximum are as follows:
1. Uses of files as command scripts, or for configuration or control, are exempt. For example, it is not required that *sh* be able to read an arbitrarily large **.profile**.
2. Shell input and output redirection are exempt. For example, it is not required that the redirections *sum < file* or *echo foo > file* succeed for an arbitrarily large existing file.

72302

1.6 Built-In Utilities

72303

72304

72305

72306

72307

72308

Any of the standard utilities may be implemented as regular built-in utilities within the command language interpreter. This is usually done to increase the performance of frequently used utilities or to achieve functionality that would be more difficult in a separate environment. The utilities named in Table 1-5 are frequently provided in built-in form. All of the utilities named in the table have special properties in terms of command search order within the shell, as described in Section 2.9.1.1 (on page 2317).

72309

Table 1-5 Regular Built-In Utilities

72310	<i>alias</i>	<i>false</i>	<i>jobs</i>	<i>read</i>	<i>wait</i>
72311	<i>bg</i>	<i>fc</i>	<i>kill</i>	<i>true</i>	
72312	<i>cd</i>	<i>fg</i>	<i>newgrp</i>	<i>umask</i>	
72313	<i>command</i>	<i>getopts</i>	<i>pwd</i>	<i>unalias</i>	

72314

72315

72316

72317

72318

However, all of the standard utilities, including the regular built-ins in the table, but not the special built-ins described in Section 2.14 (on page 2334), shall be implemented in a manner so that they can be accessed via the *exec* family of functions as defined in the System Interfaces volume of POSIX.1-2008 and can be invoked directly by those standard utilities that require it (*env*, *find*, *nice*, *nohup*, *time*, *xargs*).

Shell Command Language



This chapter contains the definition of the Shell Command Language.

2.1 Shell Introduction

The shell is a command language interpreter. This chapter describes the syntax of that command language as it is used by the *sh* utility and the *system()* and *popen()* functions defined in the System Interfaces volume of POSIX.1-2008.

The shell operates according to the following general overview of operations. The specific details are included in the cited sections of this chapter.

1. The shell reads its input from a file (see *sh*), from the *-c* option or from the *system()* and *popen()* functions defined in the System Interfaces volume of POSIX.1-2008. If the first line of a file of shell commands starts with the characters "#!", the results are unspecified.
2. The shell breaks the input into tokens: words and operators; see [Section 2.3](#) (on page 2299).
3. The shell parses the input into simple commands (see [Section 2.9.1](#), on page 2316) and compound commands (see [Section 2.9.4](#), on page 2321).
4. The shell performs various expansions (separately) on different parts of each command, resulting in a list of pathnames and fields to be treated as a command and arguments; see [Section 2.6](#) (on page 2305).
5. The shell performs redirection (see [Section 2.7](#), on page 2312) and removes redirection operators and their operands from the parameter list.
6. The shell executes a function (see [Section 2.9.5](#), on page 2324), built-in (see [Section 2.14](#), on page 2334), executable file, or script, giving the names of the arguments as positional parameters numbered 1 to *n*, and the name of the command (or in the case of a function within a script, the name of the script) as the positional parameter numbered 0 (see [Section 2.9.1.1](#), on page 2317).
7. The shell optionally waits for the command to complete and collects the exit status (see [Section 2.8.2](#), on page 2315).

2.2 Quoting

Quoting is used to remove the special meaning of certain characters or words to the shell. Quoting can be used to preserve the literal meaning of the special characters in the next paragraph, prevent reserved words from being recognized as such, and prevent parameter expansion and command substitution within here-document processing (see [Section 2.7.4](#), on page 2313).

The application shall quote the following characters if they are to represent themselves:

| & ; < > () \$ ` \ " ' <space> <tab> <newline>

and the following may need to be quoted under certain circumstances. That is, these characters may be special depending on conditions described elsewhere in this volume of POSIX.1-2008:

* ? [# ~ = %

The various quoting mechanisms are the escape character, single-quotes, and double-quotes. The here-document represents another form of quoting; see [Section 2.7.4](#) (on page 2313).

2.2.1 Escape Character (Backslash)

A <backslash> that is not quoted shall preserve the literal value of the following character, with the exception of a <newline>. If a <newline> follows the <backslash>, the shell shall interpret this as line continuation. The <backslash> and <newline> shall be removed before splitting the input into tokens. Since the escaped <newline> is removed entirely from the input and is not replaced by any white space, it cannot serve as a token separator.

2.2.2 Single-Quotes

Enclosing characters in single-quotes (' ') shall preserve the literal value of each character within the single-quotes. A single-quote cannot occur within single-quotes.

2.2.3 Double-Quotes

Enclosing characters in double-quotes (" ") shall preserve the literal value of all characters within the double-quotes, with the exception of the characters backquote, <dollar-sign>, and <backslash>, as follows:

\$ The <dollar-sign> shall retain its special meaning introducing parameter expansion (see [Section 2.6.2](#), on page 2306), a form of command substitution (see [Section 2.6.3](#), on page 2309), and arithmetic expansion (see [Section 2.6.4](#), on page 2310).

The input characters within the quoted string that are also enclosed between "\$ (" and the matching ') ' shall not be affected by the double-quotes, but rather shall define that command whose output replaces the "\$ (...) " when the word is expanded. The tokenizing rules in [Section 2.3](#) (on page 2299), not including the alias substitutions in [Section 2.3.1](#) (on page 2300), shall be applied recursively to find the matching ') '.

Within the string of characters from an enclosed "\$ { " to the matching ' } ' , an even number of unescaped double-quotes or single-quotes, if any, shall occur. A preceding <backslash> character shall be used to escape a literal ' ' or ' } ' . The rule in [Section 2.6.2](#) (on page 2306) shall be used to determine the matching ' } ' .

‘ The backquote shall retain its special meaning introducing the other form of command substitution (see [Section 2.6.3](#), on page 2309). The portion of the quoted string from the initial backquote and the characters up to the next backquote that is not preceded by a <backslash>, having escape characters removed, defines that command whose output replaces " ` . . . ` " when the word is expanded. Either of the following cases produces undefined results:

- A single-quoted or double-quoted string that begins, but does not end, within the " ` . . . ` " sequence
- A " ` . . . ` " sequence that begins, but does not end, within the same double-quoted string

\ The <backslash> shall retain its special meaning as an escape character (see [Section 2.2.1](#), on page 2298) only when followed by one of the following characters when considered special:

\$ ` " \ <newline>

The application shall ensure that a double-quote is preceded by a <backslash> to be included within double-quotes. The parameter '@' has special meaning inside double-quotes and is described in [Section 2.5.2](#) (on page 2302).

2.3 Token Recognition

The shell shall read its input in terms of lines from a file, from a terminal in the case of an interactive shell, or from a string in the case of *sh -c* or *system()*. The input lines can be of unlimited length. These lines shall be parsed using two major modes: ordinary token recognition and processing of here-documents.

When an **io_here** token has been recognized by the grammar (see [Section 2.10](#), on page 2325), one or more of the subsequent lines immediately following the next **NEWLINE** token form the body of one or more here-documents and shall be parsed according to the rules of [Section 2.7.4](#) (on page 2313).

When it is not processing an **io_here**, the shell shall break its input into tokens by applying the first applicable rule below to the next character in its input. The token shall be from the current position in the input until a token is delimited according to one of the rules below; the characters forming the token are exactly those in the input, including any quoting characters. If it is indicated that a token is delimited, and no characters have been included in a token, processing shall continue until an actual token is delimited.

1. If the end of input is recognized, the current token shall be delimited. If there is no current token, the end-of-input indicator shall be returned as the token.
2. If the previous character was used as part of an operator and the current character is not quoted and can be used with the current characters to form an operator, it shall be used as part of that (operator) token.
3. If the previous character was used as part of an operator and the current character cannot be used with the current characters to form an operator, the operator containing the previous character shall be delimited.
4. If the current character is <backslash>, single-quote, or double-quote and it is not quoted, it shall affect quoting for subsequent characters up to the end of the quoted text. The rules for quoting are as described in [Section 2.2](#) (on page 2298). During token recognition no substitutions shall be actually performed, and the result token shall contain exactly the characters that appear in the input (except for <newline> joining), unmodified, including

- any embedded or enclosing quotes or substitution operators, between the <quotation-mark> and the end of the quoted text. The token shall not be delimited by the end of the quoted field.
5. If the current character is an unquoted ' \$ ' or ' ` ', the shell shall identify the start of any candidates for parameter expansion (Section 2.6.2, on page 2306), command substitution (Section 2.6.3, on page 2309), or arithmetic expansion (Section 2.6.4, on page 2310) from their introductory unquoted character sequences: ' \$ ' or "\$ { ", "\$ (" or ' ` ', and "\$ ((", respectively. The shell shall read sufficient input to determine the end of the unit to be expanded (as explained in the cited sections). While processing the characters, if instances of expansions or quoting are found nested within the substitution, the shell shall recursively process them in the manner specified for the construct that is found. The characters found from the beginning of the substitution to its end, allowing for any recursion necessary to recognize embedded constructs, shall be included unmodified in the result token, including any embedded or enclosing substitution operators or quotes. The token shall not be delimited by the end of the substitution.
 6. If the current character is not quoted and can be used as the first character of a new operator, the current token (if any) shall be delimited. The current character shall be used as the beginning of the next (operator) token.
 7. If the current character is an unquoted <newline>, the current token shall be delimited.
 8. If the current character is an unquoted <blank>, any token containing the previous character is delimited and the current character shall be discarded.
 9. If the previous character was part of a word, the current character shall be appended to that word.
 10. If the current character is a ' # ', it and all subsequent characters up to, but excluding, the next <newline> shall be discarded as a comment. The <newline> that ends the line is not considered part of the comment.
 11. The current character is used as the start of a new word.

Once a token is delimited, it is categorized as required by the grammar in Section 2.10 (on page 2325).

2.3.1 Alias Substitution

After a token has been delimited, but before applying the grammatical rules in Section 2.10 (on page 2325), a resulting word that is identified to be the command name word of a simple command shall be examined to determine whether it is an unquoted, valid alias name. However, reserved words in correct grammatical context shall not be candidates for alias substitution. A valid alias name (see XBD Section 3.10, on page 34) shall be one that has been defined by the *alias* utility and not subsequently undefined using *unalias*. Implementations also may provide predefined valid aliases that are in effect when the shell is invoked. To prevent infinite loops in recursive aliasing, if the shell is not currently processing an alias of the same name, the word shall be replaced by the value of the alias; otherwise, it shall not be replaced.

If the value of the alias replacing the word ends in a <blank>, the shell shall check the next command word for alias substitution; this process shall continue until a word is found that is not a valid alias or an alias value does not end in a <blank>.

When used as specified by this volume of POSIX.1-2008, alias definitions shall not be inherited by separate invocations of the shell or by the utility execution environments invoked by the shell; see Section 2.12 (on page 2331).

2.4 Reserved Words

Reserved words are words that have special meaning to the shell; see [Section 2.9](#) (on page 2316). The following words shall be recognized as reserved words:

!	do	esac	in
{	done	fi	then
}	elif	for	until
case	else	if	while

This recognition shall only occur when none of the characters is quoted and when the word is used as:

- The first word of a command
- The first word following one of the reserved words other than **case**, **for**, or **in**
- The third word in a **case** command (only **in** is valid in this case)
- The third word in a **for** command (only **in** and **do** are valid in this case)

See the grammar in [Section 2.10](#) (on page 2325).

The following words may be recognized as reserved words on some implementations (when none of the characters are quoted), causing unspecified results:

[	]	function	select

Words that are the concatenation of a name and a <colon> (':') are reserved; their use produces unspecified results.

2.5 Parameters and Variables

A parameter can be denoted by a name, a number, or one of the special characters listed in [Section 2.5.2](#) (on page 2302). A variable is a parameter denoted by a name.

A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can only be unset by using the *unset* special built-in command.

2.5.1 Positional Parameters

A positional parameter is a parameter denoted by the decimal value represented by one or more digits, other than the single digit 0. The digits denoting the positional parameters shall always be interpreted as a decimal value, even if there is a leading zero. When a positional parameter with more than one digit is specified, the application shall enclose the digits in braces (see [Section 2.6.2](#), on page 2306). Positional parameters are initially assigned when the shell is invoked (see *sh*), temporarily replaced when a shell function is invoked (see [Section 2.9.5](#), on page 2324), and can be reassigned with the *set* special built-in command.

2.5.2 Special Parameters

Listed below are the special parameters and the values to which they shall expand. Only the values of the special parameters are listed; see [Section 2.6](#) (on page 2305) for a detailed summary of all the stages involved in expanding words.

- @ Expands to the positional parameters, starting from one. When the expansion occurs within double-quotes, and where field splitting (see [Section 2.6.5](#), on page 2311) is performed, each positional parameter shall expand as a separate field, with the provision that the expansion of the first parameter shall still be joined with the beginning part of the original word (assuming that the expanded parameter was embedded within a word), and the expansion of the last parameter shall still be joined with the last part of the original word. If there are no positional parameters, the expansion of ' @ ' shall generate zero fields, even when ' @ ' is double-quoted.
- * Expands to the positional parameters, starting from one. When the expansion occurs within a double-quoted string (see [Section 2.2.3](#), on page 2298), it shall expand to a single field with the value of each parameter separated by the first character of the *IFS* variable, or by a <space> if *IFS* is unset. If *IFS* is set to a null string, this is not equivalent to unsetting it; its first character does not exist, so the parameter values are concatenated.
- # Expands to the decimal number of positional parameters. The command name (parameter 0) shall not be counted in the number given by ' # ' because it is a special parameter, not a positional parameter.
- ? Expands to the decimal exit status of the most recent pipeline (see [Section 2.9.2](#), on page 2318).
- (Hyphen.) Expands to the current option flags (the single-letter option names concatenated into a string) as specified on invocation, by the *set* special built-in command, or implicitly by the shell.
- \$ Expands to the decimal process ID of the invoked shell. In a subshell (see [Section 2.12](#), on page 2331), ' \$ ' shall expand to the same value as that of the current shell.
- ! Expands to the decimal process ID of the most recent background command (see [Section 2.9.3](#), on page 2319) executed from the current shell. (For example, background commands executed from subshells do not affect the value of " \$! " in the current shell environment.) For a pipeline, the process ID is that of the last command in the pipeline.
- 0 (Zero.) Expands to the name of the shell or shell script. See *sh* (on page 3163) for a detailed description of how this name is derived.

See the description of the *IFS* variable in [Section 2.5.3](#).

2.5.3 Shell Variables

Variables shall be initialized from the environment (as defined by XBD [Chapter 8](#) (on page 173) and the *exec* function in the System Interfaces volume of POSIX.1-2008) and can be given new values with variable assignment commands. If a variable is initialized from the environment, it shall be marked for export immediately; see the *export* special built-in. New variables can be defined and initialized with variable assignments, with the *read* or *getopts* utilities, with the *name* parameter in a **for** loop, with the $\${name=word}$ expansion, or with other mechanisms provided as implementation extensions.

72549		The following variables shall affect the execution of the shell:
72550	UP XSI	ENV The processing of the <i>ENV</i> shell variable shall be supported on all XSI-conformant systems or if the system supports the User Portability Utilities option.
72551		
72552		
72553		This variable, when and only when an interactive shell is invoked, shall be subjected to parameter expansion (see Section 2.6.2 , on page 2306) by the shell and the resulting value shall be used as a pathname of a file containing shell commands to execute in the current environment. The file need not be executable. If the expanded value of <i>ENV</i> is not an absolute pathname, the results are unspecified. <i>ENV</i> shall be ignored if the user's real and effective user IDs or real and effective group IDs are different.
72554		
72555		
72556		
72557		
72558		
72559		
72560	<i>HOME</i>	The pathname of the user's home directory. The contents of <i>HOME</i> are used in tilde expansion (see Section 2.6.1 , on page 2305).
72561		
72562	<i>IFS</i>	A string treated as a list of characters that is used for field splitting and to split lines into fields with the <i>read</i> command.
72563		
72564		If <i>IFS</i> is not set, it shall behave as normal for an unset variable, except that field splitting by the shell and line splitting by the <i>read</i> command shall be performed as if the value of <i>IFS</i> is <space><tab><newline>; see Section 2.6.5 (on page 2311).
72565		
72566		
72567		
72568		Implementations may ignore the value of <i>IFS</i> in the environment, or the absence of <i>IFS</i> from the environment, at the time the shell is invoked, in which case the shell shall set <i>IFS</i> to <space><tab><newline> when it is invoked.
72569		
72570		
72571	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
72572		
72573		
72574		
72575	<i>LC_ALL</i>	The value of this variable overrides the <i>LC_*</i> variables and <i>LANG</i> , as described in XBD Chapter 8 (on page 173).
72576		
72577	<i>LC_COLLATE</i>	Determine the behavior of range expressions, equivalence classes, and multi-character collating elements within pattern matching.
72578		
72579	<i>LC_CTYPE</i>	Determine the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters), which characters are defined as letters (character class alpha) and <blank> characters (character class blank), and the behavior of character classes within pattern matching. Changing the value of <i>LC_CTYPE</i> after the shell has started shall not affect the lexical processing of shell commands in the current shell execution environment or its subshells. Invoking a shell script or performing <i>exec sh</i> subjects the new shell to the changes in <i>LC_CTYPE</i> .
72580		
72581		
72582		
72583		
72584		
72585		
72586		
72587	<i>LC_MESSAGES</i>	Determine the language in which messages should be written.
72588	<i>LINENO</i>	Set by the shell to a decimal number representing the current sequential line number (numbered starting with 1) within a script or function before it executes each command. If the user unsets or resets <i>LINENO</i> , the variable may lose its special meaning for the life of the shell. If the shell is not currently executing a script or function, the value of <i>LINENO</i> is unspecified. This volume of POSIX.1-2008 specifies the effects of the variable only for systems supporting the User Portability Utilities option.
72589		
72590		
72591		
72592		
72593		
72594		

72595	XSI	NLSPATH	Determine the location of message catalogs for the processing of
72596			LC_MESSAGES.
72597		PATH	A string formatted as described in XBD Chapter 8 (on page 173), used to effect
72598			command interpretation; see Section 2.9.1.1 (on page 2317).
72599		PPID	Set by the shell to the decimal value of its parent process ID during
72600			initialization of the shell. In a subshell (see Section 2.12, on page 2331), <i>PPID</i>
72601			shall be set to the same value as that of the parent of the current shell. For
72602			example, <i>echo \$PPID</i> and (<i>echo \$PPID</i>) would produce the same value. This
72603			volume of POSIX.1-2008 specifies the effects of the variable only for systems
72604			supporting the User Portability Utilities option.
72605		PS1	Each time an interactive shell is ready to read a command, the value of this
72606			variable shall be subjected to parameter expansion and written to standard
72607			error. The default value shall be "\$ ". For users who have specific additional
72608			implementation-defined privileges, the default may be another,
72609			implementation-defined value. The shell shall replace each instance of the
72610			character '!' in <i>PS1</i> with the history file number of the next command to be
72611			typed. Escaping the '!' with another '!' (that is, "!!") shall place the literal
72612			character '!' in the prompt. This volume of POSIX.1-2008 specifies the effects
72613			of the variable only for systems supporting the User Portability Utilities
72614			option.
72615		PS2	Each time the user enters a <newline> prior to completing a command line in
72616			an interactive shell, the value of this variable shall be subjected to parameter
72617			expansion and written to standard error. The default value is "> ". This
72618			volume of POSIX.1-2008 specifies the effects of the variable only for systems
72619			supporting the User Portability Utilities option.
72620		PS4	When an execution trace (<i>set -x</i>) is being performed in an interactive shell,
72621			before each line in the execution trace, the value of this variable shall be
72622			subjected to parameter expansion and written to standard error. The default
72623			value is "+ ". This volume of POSIX.1-2008 specifies the effects of the
72624			variable only for systems supporting the User Portability Utilities option.
72625		PWD	Set by the shell and by the <i>cd</i> utility. In the shell the value shall be initialized
72626			from the environment as follows. If a value for <i>PWD</i> is passed to the shell in
72627			the environment when it is executed, the value is an absolute pathname of the
72628			current working directory that is no longer than {PATH_MAX} bytes including
72629			the terminating null byte, and the value does not contain any components that
72630			are dot or dot-dot, then the shell shall set <i>PWD</i> to the value from the
72631			environment. Otherwise, if a value for <i>PWD</i> is passed to the shell in the
72632			environment when it is executed, the value is an absolute pathname of the
72633			current working directory, and the value does not contain any components
72634			that are dot or dot-dot, then it is unspecified whether the shell sets <i>PWD</i> to the
72635			value from the environment or sets <i>PWD</i> to the pathname that would be
72636			output by <i>pwd -P</i> . Otherwise, the <i>sh</i> utility sets <i>PWD</i> to the pathname that
72637			would be output by <i>pwd -P</i> . In cases where <i>PWD</i> is set to the value from the
72638			environment, the value can contain components that refer to files of type
72639			symbolic link. In cases where <i>PWD</i> is set to the pathname that would be
72640			output by <i>pwd -P</i> , if there is insufficient permission on the current working
72641			directory, or on any parent of that directory, to determine what that pathname
72642			would be, the value of <i>PWD</i> is unspecified. Assignments to this variable may
72643			be ignored. If an application sets or unsets the value of <i>PWD</i> , the behaviors of
72644			the <i>cd</i> and <i>pwd</i> utilities are unspecified.

2.6 Word Expansions

This section describes the various expansions that are performed on words. Not all expansions are performed on every word, as explained in the following sections.

Tilde expansions, parameter expansions, command substitutions, arithmetic expansions, and quote removals that occur within a single word expand to a single field. It is only field splitting or pathname expansion that can create multiple fields from a single word. The single exception to this rule is the expansion of the special parameter '@' within double-quotes, as described in [Section 2.5.2](#) (on page 2302).

The order of word expansion shall be as follows:

1. Tilde expansion (see [Section 2.6.1](#)), parameter expansion (see [Section 2.6.2](#), on page 2306), command substitution (see [Section 2.6.3](#), on page 2309), and arithmetic expansion (see [Section 2.6.4](#), on page 2310) shall be performed, beginning to end. See item 5 in [Section 2.3](#) (on page 2299).
2. Field splitting (see [Section 2.6.5](#), on page 2311) shall be performed on the portions of the fields generated by step 1, unless *IFS* is null.
3. Pathname expansion (see [Section 2.6.6](#), on page 2311) shall be performed, unless *set -f* is in effect.
4. Quote removal (see [Section 2.6.7](#), on page 2311) shall always be performed last.

The expansions described in this section shall occur in the same shell environment as that in which the command is executed.

If the complete expansion appropriate for a word results in an empty field, that empty field shall be deleted from the list of fields that form the completely expanded command, unless the original word contained single-quote or double-quote characters.

The '\$' character is used to introduce parameter expansion, command substitution, or arithmetic evaluation. If an unquoted '\$' is followed by a character that is either not numeric, the name of one of the special parameters (see [Section 2.5.2](#), on page 2302), a valid first character of a variable name, a <left-curly-bracket> ('{') or a <left-parenthesis>, the result is unspecified.

2.6.1 Tilde Expansion

A "tilde-prefix" consists of an unquoted <tilde> character at the beginning of a word, followed by all of the characters preceding the first unquoted <slash> in the word, or all the characters in the word if there is no <slash>. In an assignment (see [XBD Section 4.22](#), on page 118), multiple tilde-prefixes can be used: at the beginning of the word (that is, following the <equals-sign> of the assignment), following any unquoted <colon>, or both. A tilde-prefix in an assignment is terminated by the first unquoted <colon> or <slash>. If none of the characters in the tilde-prefix are quoted, the characters in the tilde-prefix following the <tilde> are treated as a possible login name from the user database. A portable login name cannot contain characters outside the set given in the description of the *LOGNAME* environment variable in [XBD Section 8.3](#) (on page 177). If the login name is null (that is, the tilde-prefix contains only the tilde), the tilde-prefix is replaced by the value of the variable *HOME*. If *HOME* is unset, the results are unspecified. Otherwise, the tilde-prefix shall be replaced by a pathname of the initial working directory associated with the login name obtained using the *getpwnam()* function as defined in the System Interfaces volume of POSIX.1-2008. If the system does not recognize the login name, the results are undefined.

The pathname resulting from tilde expansion shall be treated as if quoted to prevent it being

72689 altered by field splitting and pathname expansion.

72690 2.6.2 Parameter Expansion

72691 The format for parameter expansion is as follows:

72692 `${expression}`

72693 where *expression* consists of all characters until the matching `'}'`. Any `'}'` escaped by a
72694 `<backslash>` or within a quoted string, and characters in embedded arithmetic expansions,
72695 command substitutions, and variable expansions, shall not be examined in determining the
72696 matching `'}'`.

72697 The simplest form for parameter expansion is:

72698 `${parameter}`

72699 The value, if any, of *parameter* shall be substituted.

72700 The parameter name or symbol can be enclosed in braces, which are optional except for
72701 positional parameters with more than one digit or when *parameter* is followed by a character that
72702 could be interpreted as part of the name. The matching closing brace shall be determined by
72703 counting brace levels, skipping over enclosed quoted strings, and command substitutions.

72704 If the parameter name or symbol is not enclosed in braces, the expansion shall use the longest
72705 valid name (see XBD Section 3.230, on page 70), whether or not the symbol represented by that
72706 name exists.

72707 If a parameter expansion occurs inside double-quotes:

- 72708 • Pathname expansion shall not be performed on the results of the expansion.
- 72709 • Field splitting shall not be performed on the results of the expansion, with the exception of
72710 `'@'`; see Section 2.5.2 (on page 2302).

72711 In addition, a parameter expansion can be modified by using one of the following formats. In
72712 each case that a value of *word* is needed (based on the state of *parameter*, as described below),
72713 *word* shall be subjected to tilde expansion, parameter expansion, command substitution, and
72714 arithmetic expansion. If *word* is not needed, it shall not be expanded. The `'}'` character that
72715 delimits the following parameter expansion modifications shall be determined as described
72716 previously in this section and in Section 2.2.3 (on page 2298). (For example, `${foo-bar}xyz`
72717 would result in the expansion of `foo` followed by the string `xyz` if `foo` is set, else the string
72718 `"barxyz"`).

72719 `${parameter:-word}` **Use Default Values.** If *parameter* is unset or null, the expansion of *word*
72720 shall be substituted; otherwise, the value of *parameter* shall be substituted.

72721 `${parameter:=word}` **Assign Default Values.** If *parameter* is unset or null, the expansion of
72722 *word* shall be assigned to *parameter*. In all cases, the final value of
72723 *parameter* shall be substituted. Only variables, not positional parameters
72724 or special parameters, can be assigned in this way.

72725 `${parameter:?[word]}` **Indicate Error if Null or Unset.** If *parameter* is unset or null, the
72726 expansion of *word* (or a message indicating it is unset if *word* is omitted)
72727 shall be written to standard error and the shell exits with a non-zero exit
72728 status. Otherwise, the value of *parameter* shall be substituted. An
72729 interactive shell need not exit.

Shell Command Language

Word Expansions

Use Alternative Value. If *parameter* is unset or null, null shall be substituted; otherwise, the expansion of *word* shall be substituted.

In the parameter expansions shown previously, use of the <colon> in the format shall result in a test for a parameter that is unset or null; omission of the <colon> shall result in a test for a parameter that is only unset. The following table summarizes the effect of the <colon>:

	<i>parameter</i> Set and Not Null	<i>parameter</i> Set But Null	<i>parameter</i> Unset
<code>\${parameter:-word}</code>	substitute <i>parameter</i>	substitute <i>word</i>	substitute <i>word</i>
<code>\${parameter-word}</code>	substitute <i>parameter</i>	substitute null	substitute <i>word</i>
<code>\${parameter:=word}</code>	substitute <i>parameter</i>	assign <i>word</i>	assign <i>word</i>
<code>\${parameter=word}</code>	substitute <i>parameter</i>	substitute null	assign <i>word</i>
<code>\${parameter:?word}</code>	substitute <i>parameter</i>	error, exit	error, exit
<code>\${parameter?word}</code>	substitute <i>parameter</i>	substitute null	error, exit
<code>\${parameter:+word}</code>	substitute <i>word</i>	substitute null	substitute null
<code>\${parameter+word}</code>	substitute <i>word</i>	substitute <i>word</i>	substitute null

In all cases shown with “substitute”, the expression is replaced with the value shown. In all cases shown with “assign”, *parameter* is assigned that value, which also replaces the expression.

String Length. The length in characters of the value of *parameter* shall be substituted. If *parameter* is '*' or '@', the result of the expansion is unspecified.

The following four varieties of parameter expansion provide for substring processing. In each case, pattern matching notation (see Section 2.13, on page 2332), rather than regular expression notation, shall be used to evaluate the patterns. If *parameter* is '*' or '@', the result of the expansion is unspecified. Enclosing the full parameter expansion string in double-quotes shall not cause the following four varieties of pattern characters to be quoted, whereas quoting characters within the braces shall have this effect.

Remove Smallest Suffix Pattern. The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the smallest portion of the suffix matched by the *pattern* deleted.

Remove Largest Suffix Pattern. The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the largest portion of the suffix matched by the *pattern* deleted.

Remove Smallest Prefix Pattern. The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the smallest portion of the prefix matched by the *pattern* deleted.

Remove Largest Prefix Pattern. The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the largest portion of the prefix matched by the *pattern* deleted.

Examples

```
${parameter:-word}
```

In this example, *ls* is executed only if *x* is null or unset. (The *\$(ls)* command substitution notation is explained in [Section 2.6.3](#) (on page 2309).)

```
${x:-$(ls)}
```

```
${parameter:=word}
```

```
unset X
```

```
echo ${X:=abc}
```

```
abc
```

```
${parameter:?word}
```

```
unset posix
```

```
echo ${posix:?}
```

```
sh: posix: parameter null or not set
```

```
${parameter:+word}
```

```
set a b c
```

```
echo ${3:+posix}
```

```
posix
```

```
${#parameter}
```

```
HOME=/usr/posix
```

```
echo ${#HOME}
```

```
10
```

```
${parameter%word}
```

```
x=file.c
```

```
echo ${x%.c}.o
```

```
file.o
```

```
${parameter%%word}
```

```
x=posix/src/std
```

```
echo ${x%%/*}
```

```
posix
```

```
${parameter#word}
```

```
x=$HOME/src/cmd
```

```
echo ${x#$HOME}
```

```
/src/cmd
```

```
${parameter##word}
```

```
x=/one/two/three
```

```
echo ${x##*/}
```

```
three
```

The double-quoting of patterns is different depending on where the double-quotes are placed:

"\${x#*}" The <asterisk> is a pattern character.

\${x#"*"} The literal <asterisk> is quoted and not special.

2.6.3 Command Substitution

Command substitution allows the output of a command to be substituted in place of the command name itself. Command substitution shall occur when the command is enclosed as follows:

`$(command)`

or (backquoted version):

``command``

The shell shall expand the command substitution by executing *command* in a subshell environment (see [Section 2.12](#), on page 2331) and replacing the command substitution (the text of *command* plus the enclosing "`$ (`" or backquotes) with the standard output of the command, removing sequences of one or more `<newline>` characters at the end of the substitution. Embedded `<newline>` characters before the end of the output shall not be removed; however, they may be treated as field delimiters and eliminated during field splitting, depending on the value of *IFS* and quoting that is in effect. If the output contains any null bytes, the behavior is unspecified.

Within the backquoted style of command substitution, `<backslash>` shall retain its literal meaning, except when followed by: `'$'`, `'`'`, or `<backslash>`. The search for the matching backquote shall be satisfied by the first unquoted non-escaped backquote; during this search, if a non-escaped backquote is encountered within a shell comment, a here-document, an embedded command substitution of the `$(command)` form, or a quoted string, undefined results occur. A single-quoted or double-quoted string that begins, but does not end, within the `"`...`"` sequence produces undefined results.

With the `$(command)` form, all characters following the open parenthesis to the matching closing parenthesis constitute the *command*. Any valid shell script can be used for *command*, except a script consisting solely of redirections which produces unspecified results.

The results of command substitution shall not be processed for further tilde expansion, parameter expansion, command substitution, or arithmetic expansion. If a command substitution occurs inside double-quotes, field splitting and pathname expansion shall not be performed on the results of the substitution.

Command substitution can be nested. To specify nesting within the backquoted version, the application shall precede the inner backquotes with `<backslash>` characters; for example:

`\`command\``

If the command substitution consists of a single subshell, such as:

`$( (command) )`

a conforming application shall separate the `"$ ("` and `' ('` into two tokens (that is, separate them with white space). This is required to avoid any ambiguities with arithmetic expansion.

2.6.4 Arithmetic Expansion

Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and substituting its value. The format for arithmetic expansion shall be as follows:

```
$ ( (expression) )
```

The expression shall be treated as if it were in double-quotes, except that a double-quote inside the expression is not treated specially. The shell shall expand all tokens in the expression for parameter expansion, command substitution, and quote removal.

Next, the shell shall treat this as an arithmetic expression and substitute the value of the expression. The arithmetic expression shall be processed according to the rules given in [Section 1.1.2.1](#) (on page 2283), with the following exceptions:

- Only signed long integer arithmetic is required.
- Only the decimal-constant, octal-constant, and hexadecimal-constant constants specified in the ISO C standard, Section 6.4.4.1 are required to be recognized as constants.
- The `sizeof()` operator and the prefix and postfix `"++"` and `"--"` operators are not required.
- Selection, iteration, and jump statements are not supported.

All changes to variables in an arithmetic expression shall be in effect after the arithmetic expansion, as in the parameter expansion `"${x=value}"`.

If the shell variable `x` contains a value that forms a valid integer constant, then the arithmetic expansions `"$ ( (x) )"` and `"$ ( ($x) )"` shall return the same value.

As an extension, the shell may recognize arithmetic expressions beyond those listed. The shell may use a signed integer type with a rank larger than the rank of **signed long**. The shell may use a real-floating type instead of **signed long** as long as it does not affect the results in cases where there is no overflow. If the expression is invalid, the expansion fails and the shell shall write a message to standard error indicating the failure.

Examples

A simple example using arithmetic expansion:

```
# repeat a command 100 times
x=100
while [ $x -gt 0 ]
do
    command
    x=$((x-1))
done
```

2.6.5 Field Splitting

After parameter expansion (Section 2.6.2, on page 2306), command substitution (Section 2.6.3, on page 2309), and arithmetic expansion (Section 2.6.4, on page 2310), the shell shall scan the results of expansions and substitutions that did not occur in double-quotes for field splitting and multiple fields can result.

The shell shall treat each character of the *IFS* as a delimiter and use the delimiters as field terminators to split the results of parameter expansion and command substitution into fields.

1. If the value of *IFS* is a <space>, <tab>, and <newline>, or if it is unset, any sequence of <space>, <tab>, or <newline> characters at the beginning or end of the input shall be ignored and any sequence of those characters within the input shall delimit a field. For example, the input:

```
<newline><space><tab>foo<tab><tab>bar<space>
```

yields two fields, **foo** and **bar**.

2. If the value of *IFS* is null, no field splitting shall be performed.
3. Otherwise, the following rules shall be applied in sequence. The term “*IFS* white space” is used to mean any sequence (zero or more instances) of white-space characters that are in the *IFS* value (for example, if *IFS* contains <space>/<comma>/<tab>, any sequence of <space> and <tab> characters is considered *IFS* white space).
 - a. *IFS* white space shall be ignored at the beginning and end of the input.
 - b. Each occurrence in the input of an *IFS* character that is not *IFS* white space, along with any adjacent *IFS* white space, shall delimit a field, as described previously.
 - c. Non-zero-length *IFS* white space shall delimit a field.

2.6.6 Pathname Expansion

After field splitting, if *set -f* is not in effect, each field in the resulting command line shall be expanded using the algorithm described in Section 2.13 (on page 2332), qualified by the rules in Section 2.13.3 (on page 2333).

2.6.7 Quote Removal

The quote characters (<backslash>, single-quote, and double-quote) that were present in the original word shall be removed unless they have themselves been quoted.

2.7 Redirection

Redirection is used to open and close files for the current shell execution environment (see [Section 2.12](#), on page 2331) or for any command. Redirection operators can be used with numbers representing file descriptors (see [XBD Section 3.166](#), on page 60) as described below.

The overall format used for redirection is:

```
[n] redir-op word
```

The number *n* is an optional decimal number designating the file descriptor number; the application shall ensure it is delimited from any preceding text and immediately precede the redirection operator *redir-op*. If *n* is quoted, the number shall not be recognized as part of the redirection expression. For example:

```
echo \2>a
```

writes the character 2 into file **a**. If any part of *redir-op* is quoted, no redirection expression is recognized. For example:

```
echo 2\>a
```

writes the characters 2>a to standard output. The optional number, redirection operator, and *word* shall not appear in the arguments provided to the command to be executed (if any).

Open files are represented by decimal numbers starting with zero. The largest possible value is implementation-defined; however, all implementations shall support at least 0 to 9, inclusive, for use by the application. These numbers are called "file descriptors". The values 0, 1, and 2 have special meaning and conventional uses and are implied by certain redirection operations; they are referred to as *standard input*, *standard output*, and *standard error*, respectively. Programs usually take their input from standard input, and write output on standard output. Error messages are usually written on standard error. The redirection operators can be preceded by one or more digits (with no intervening <blank> characters allowed) to designate the file descriptor number.

If the redirection operator is "<<" or "<<-", the word that follows the redirection operator shall be subjected to quote removal; it is unspecified whether any of the other expansions occur. For the other redirection operators, the word that follows the redirection operator shall be subjected to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal. Pathname expansion shall not be performed on the word by a non-interactive shell; an interactive shell may perform it, but shall do so only when the expansion would result in one word.

If more than one redirection operator is specified with a command, the order of evaluation is from beginning to end.

A failure to open or create a file shall cause a redirection to fail.

2.7.1 Redirecting Input

Input redirection shall cause the file whose name results from the expansion of *word* to be opened for reading on the designated file descriptor, or standard input if the file descriptor is not specified.

The general format for redirecting input is:

```
[n] <word
```

where the optional *n* represents the file descriptor number. If the number is omitted, the

72948 redirection shall refer to standard input (file descriptor 0).

72949 **2.7.2 Redirecting Output**

72950 The two general formats for redirecting output are:

72951 `[n]>word`

72952 `[n]>|word`

72953 where the optional *n* represents the file descriptor number. If the number is omitted, the
72954 redirection shall refer to standard output (file descriptor 1).

72955 Output redirection using the '`>`' format shall fail if the *noclobber* option is set (see the
72956 description of *set -C*) and the file named by the expansion of *word* exists and is a regular file.
72957 Otherwise, redirection using the '`>`' or '`>|`' formats shall cause the file whose name results
72958 from the expansion of *word* to be created and opened for output on the designated file
72959 descriptor, or standard output if none is specified. If the file does not exist, it shall be created;
72960 otherwise, it shall be truncated to be an empty file after being opened.

72961 **2.7.3 Appending Redirected Output**

72962 Appended output redirection shall cause the file whose name results from the expansion of
72963 *word* to be opened for output on the designated file descriptor. The file is opened as if the *open()*
72964 function as defined in the System Interfaces volume of POSIX.1-2008 was called with the
72965 *O_APPEND* flag. If the file does not exist, it shall be created.

72966 The general format for appending redirected output is as follows:

72967 `[n]>>word`

72968 where the optional *n* represents the file descriptor number. If the number is omitted, the
72969 redirection refers to standard output (file descriptor 1).

72970 **2.7.4 Here-Document**

72971 The redirection operators '`<<`' and '`<<-`' both allow redirection of lines contained in a shell
72972 input file, known as a "here-document", to the input of a command.

72973 The here-document shall be treated as a single word that begins after the next <newline> and
72974 continues until there is a line containing only the delimiter and a <newline>, with no <blank>
72975 characters in between. Then the next here-document starts, if there is one. The format is as
72976 follows:

72977 `[n]<<word`

72978 `here-document`

72979 `delimiter`

72980 where the optional *n* represents the file descriptor number. If the number is omitted, the here-
72981 document refers to standard input (file descriptor 0).

72982 If any character in *word* is quoted, the delimiter shall be formed by performing quote removal on
72983 *word*, and the here-document lines shall not be expanded. Otherwise, the delimiter shall be the
72984 *word* itself.

72985 If no characters in *word* are quoted, all lines of the here-document shall be expanded for
72986 parameter expansion, command substitution, and arithmetic expansion. In this case, the

72987 <backslash> in the input behaves as the <backslash> inside double-quotes (see [Section 2.2.3](#), on
 72988 page 2298). However, the double-quote character (' ') shall not be treated specially within a
 72989 here-document, except when the double-quote appears within "\$()", "'\"", or "\${ }".

72990 If the redirection symbol is "<<-", all leading <tab> characters shall be stripped from input lines
 72991 and the line containing the trailing delimiter. If more than one "<<" or "<<-" operator is
 72992 specified on a line, the here-document associated with the first operator shall be supplied first
 72993 by the application and shall be read first by the shell.

72994 When a here-document is read from a terminal device and the shell is interactive, it shall write
 72995 the contents of the variable *PS2*, processed as described in [Section 2.5.3](#) (on page 2302), to
 72996 standard error before reading each line of input until the delimiter has been recognized.

72997 Examples

72998 An example of a here-document follows:

```
72999 cat <<eof1; cat <<eof2
73000 Hi,
73001 eof1
73002 Helene.
73003 eof2
```

73004 2.7.5 Duplicating an Input File Descriptor

73005 The redirection operator:

73006 `[n]<&word`

73007 shall duplicate one input file descriptor from another, or shall close one. If *word* evaluates to one
 73008 or more digits, the file descriptor denoted by *n*, or standard input if *n* is not specified, shall be
 73009 made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a
 73010 file descriptor already open for input, a redirection error shall result; see [Section 2.8.1](#) (on page
 73011 2315). If *word* evaluates to '-', file descriptor *n*, or standard input if *n* is not specified, shall be
 73012 closed. Attempts to close a file descriptor that is not open shall not constitute an error. If *word*
 73013 evaluates to something else, the behavior is unspecified.

73014 2.7.6 Duplicating an Output File Descriptor

73015 The redirection operator:

73016 `[n]>&word`

73017 shall duplicate one output file descriptor from another, or shall close one. If *word* evaluates to
 73018 one or more digits, the file descriptor denoted by *n*, or standard output if *n* is not specified, shall
 73019 be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent
 73020 a file descriptor already open for output, a redirection error shall result; see [Section 2.8.1](#) (on
 73021 page 2315). If *word* evaluates to '-', file descriptor *n*, or standard output if *n* is not specified, is
 73022 closed. Attempts to close a file descriptor that is not open shall not constitute an error. If *word*
 73023 evaluates to something else, the behavior is unspecified.

2.7.7 Open File Descriptors for Reading and Writing

The redirection operator:

`[n] <>word`

shall cause the file whose name is the expansion of *word* to be opened for both reading and writing on the file descriptor denoted by *n*, or standard input if *n* is not specified. If the file does not exist, it shall be created.

2.8 Exit Status and Errors

2.8.1 Consequences of Shell Errors

For a non-interactive shell, an error condition encountered by a special built-in (see [Section 2.14](#), on page 2334) or other type of utility shall cause the shell to write a diagnostic message to standard error and exit as shown in the following table:

Error	Special Built-In	Other Utilities
Shell language syntax error	Shall exit	Shall exit
Utility syntax error (option or operand error)	Shall exit	Shall not exit
Redirection error	Shall exit	Shall not exit
Variable assignment error	Shall exit	Shall not exit
Expansion error	Shall exit	Shall exit
Command not found	N/A	May exit
Dot script not found	Shall exit	N/A

An expansion error is one that occurs when the shell expansions defined in [Section 2.6](#) (on page 2305) are carried out (for example, "`${x!y}`", because '`!`' is not a valid operator); an implementation may treat these as syntax errors if it is able to detect them during tokenization, rather than during expansion.

If any of the errors shown as "shall exit" or "(may) exit" occur in a subshell, the subshell shall (respectively may) exit with a non-zero status, but the script containing the subshell shall not exit because of the error.

In all of the cases shown in the table, an interactive shell shall write a diagnostic message to standard error without exiting.

2.8.2 Exit Status for Commands

Each command has an exit status that can influence the behavior of other shell commands. The exit status of commands that are not utilities is documented in this section. The exit status of the standard utilities is documented in their respective sections.

If a command is not found, the exit status shall be 127. If the command name is found, but it is not an executable utility, the exit status shall be 126. Applications that invoke utilities without using the shell should use these exit status values to report similar errors.

If a command fails during word expansion or redirection, its exit status shall be greater than zero.

Internally, for purposes of deciding whether a command exits with a non-zero exit status, the shell shall recognize the entire status value retrieved for the command by the equivalent of the `wait()` function `WEXITSTATUS` macro (as defined in the System Interfaces volume of POSIX.1-2008). When reporting the exit status with the special parameter `'?'`, the shell shall report the full eight bits of exit status available. The exit status of a command that terminated because it received a signal shall be reported as greater than 128.

2.9 Shell Commands

This section describes the basic structure of shell commands. The following command descriptions each describe a format of the command that is only used to aid the reader in recognizing the command type, and does not formally represent the syntax. Each description discusses the semantics of the command; for a formal definition of the command language, consult [Section 2.10](#) (on page 2325).

A *command* is one of the following:

- Simple command (see [Section 2.9.1](#))
- Pipeline (see [Section 2.9.2](#), on page 2318)
- List compound-list (see [Section 2.9.3](#), on page 2319)
- Compound command (see [Section 2.9.4](#), on page 2321)
- Function definition (see [Section 2.9.5](#), on page 2324)

Unless otherwise stated, the exit status of a command shall be that of the last simple command executed by the command. There shall be no limit on the size of any shell command other than that imposed by the underlying system (memory constraints, `{ARG_MAX}`, and so on).

2.9.1 Simple Commands

A “simple command” is a sequence of optional variable assignments and redirections, in any sequence, optionally followed by words and redirections, terminated by a control operator.

When a given simple command is required to be executed (that is, when any conditional construct such as an AND-OR list or a **case** statement has not bypassed the simple command), the following expansions, assignments, and redirections shall all be performed from the beginning of the command text to the end:

1. The words that are recognized as variable assignments or redirections according to [Section 2.10.2](#) (on page 2325) are saved for processing in steps 3 and 4.
2. The words that are not variable assignments or redirections shall be expanded. If any fields remain following their expansion, the first field shall be considered the command name and remaining fields are the arguments for the command.
3. Redirections shall be performed as described in [Section 2.7](#) (on page 2312).
4. Each variable assignment shall be expanded for tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal prior to assigning the value.

In the preceding list, the order of steps 3 and 4 may be reversed if no command name results from step 2 or if the command name matches the name of a special built-in utility; see [Section 2.14](#) (on page 2334).

If no command name results, variable assignments shall affect the current execution environment. Otherwise, the variable assignments shall be exported for the execution environment of the command and shall not affect the current execution environment (except for special built-ins). If any of the variable assignments attempt to assign a value to a read-only variable, a variable assignment error shall occur. See [Section 2.8.1](#) (on page 2315) for the consequences of these errors.

If there is no command name, any redirections shall be performed in a subshell environment; it is unspecified whether this subshell environment is the same one as that used for a command substitution within the command. (To affect the current execution environment, see the *exec* special built-in.) If any of the redirections performed in the current shell execution environment fail, the command shall immediately fail with an exit status greater than zero, and the shell shall write an error message indicating the failure. See [Section 2.8.1](#) (on page 2315) for the consequences of these failures on interactive and non-interactive shells.

If there is a command name, execution shall continue as described in [Section 2.9.1.1](#). If there is no command name, but the command contained a command substitution, the command shall complete with the exit status of the last command substitution performed. Otherwise, the command shall complete with a zero exit status.

2.9.1.1 Command Search and Execution

If a simple command results in a command name and an optional list of arguments, the following actions shall be performed:

1. If the command name does not contain any <slash> characters, the first successful step in the following sequence shall occur:
 - a. If the command name matches the name of a special built-in utility, that special built-in utility shall be invoked.
 - b. If the command name matches the name of a function known to this shell, the function shall be invoked as described in [Section 2.9.5](#) (on page 2324). If the implementation has provided a standard utility in the form of a function, it shall not be recognized at this point. It shall be invoked in conjunction with the path search in step 1d.
 - c. If the command name matches the name of a utility listed in the following table, that utility shall be invoked.

<i>alias</i>	<i>false</i>	<i>jobs</i>	<i>read</i>	<i>wait</i>
<i>bg</i>	<i>fc</i>	<i>kill</i>	<i>true</i>	
<i>cd</i>	<i>fg</i>	<i>newgrp</i>	<i>umask</i>	
<i>command</i>	<i>getopts</i>	<i>pwd</i>	<i>unalias</i>	

- d. Otherwise, the command shall be searched for using the *PATH* environment variable as described in [XBD Chapter 8](#) (on page 173):

- i. If the search is successful:

- a. If the system has implemented the utility as a regular built-in or as a shell function, it shall be invoked at this point in the path search.
- b. Otherwise, the shell executes the utility in a separate utility environment (see [Section 2.12](#), on page 2331) with actions equivalent to calling the *execve()* function as defined in the System Interfaces volume of POSIX.1-2008 with the *path* argument set to the pathname resulting from the search, *arg0* set to the command name, and the

remaining arguments set to the operands, if any.

If the *execve()* function fails due to an error equivalent to the [ENOEXEC] error defined in the System Interfaces volume of POSIX.1-2008, the shell shall execute a command equivalent to having a shell invoked with the pathname resulting from the search as its first operand, with any remaining arguments passed to the new shell, except that the value of "\$0" in the new shell may be set to the command name. If the executable file is not a text file, the shell may bypass this command execution. In this case, it shall write an error message, and shall return an exit status of 126.

Once a utility has been searched for and found (either as a result of this specific search or as part of an unspecified shell start-up activity), an implementation may remember its location and need not search for the utility again unless the *PATH* variable has been the subject of an assignment. If the remembered location fails for a subsequent invocation, the shell shall repeat the search to find the new location for the utility, if any.

- ii. If the search is unsuccessful, the command shall fail with an exit status of 127 and the shell shall write an error message.

- 2. If the command name contains at least one <slash>, the shell shall execute the utility in a separate utility environment with actions equivalent to calling the *execve()* function defined in the System Interfaces volume of POSIX.1-2008 with the *path* and *arg0* arguments set to the command name, and the remaining arguments set to the operands, if any.

If the *execve()* function fails due to an error equivalent to the [ENOEXEC] error, the shell shall execute a command equivalent to having a shell invoked with the command name as its first operand, with any remaining arguments passed to the new shell. If the executable file is not a text file, the shell may bypass this command execution. In this case, it shall write an error message and shall return an exit status of 126.

2.9.2 Pipelines

A *pipeline* is a sequence of one or more commands separated by the control operator '|'. The standard output of all but the last command shall be connected to the standard input of the next command.

The format for a pipeline is:

```
[!] command1 [ | command2 ... ]
```

The standard output of *command1* shall be connected to the standard input of *command2*. The standard input, standard output, or both of a command shall be considered to be assigned by the pipeline before any redirection specified by redirection operators that are part of the command (see [Section 2.7](#), on page 2312).

If the pipeline is not in the background (see [Section 2.9.3.1](#), on page 2319), the shell shall wait for the last command specified in the pipeline to complete, and may also wait for all commands to complete.

Exit Status

If the reserved word **!** does not precede the pipeline, the exit status shall be the exit status of the last command specified in the pipeline. Otherwise, the exit status shall be the logical NOT of the exit status of the last command. That is, if the last command returns zero, the exit status shall be 1; if the last command returns greater than zero, the exit status shall be zero.

2.9.3 Lists

An *AND-OR list* is a sequence of one or more pipelines separated by the operators **"&&"** and **"||"**.

A *list* is a sequence of one or more AND-OR lists separated by the operators **' ; '** and **' & '** and optionally terminated by **' ; '**, **' & '**, or **<newline>**.

The operators **"&&"** and **"||"** shall have equal precedence and shall be evaluated with left associativity. For example, both of the following commands write solely **bar** to standard output:

```
false && echo foo || echo bar
true || echo foo && echo bar
```

A **' ; '** or **<newline>** terminator shall cause the preceding AND-OR list to be executed sequentially; an **' & '** shall cause asynchronous execution of the preceding AND-OR list.

The term "compound-list" is derived from the grammar in [Section 2.10](#) (on page 2325); it is equivalent to a sequence of *lists*, separated by **<newline>** characters, that can be preceded or followed by an arbitrary number of **<newline>** characters.

Examples

The following is an example that illustrates **<newline>** characters in compound-lists:

```
while
    # a couple of <newline>s
    # a list
    date && who || ls; cat file
    # a couple of <newline>s
    # another list
    wc file > output & true
do
    # 2 lists
    ls
    cat file
done
```

2.9.3.1 Asynchronous Lists

If a command is terminated by the control operator **<ampersand>** (**' & '**), the shell shall execute the command asynchronously in a subshell. This means that the shell shall not wait for the command to finish before executing the next command.

The format for running a command in the background is:

```
command1 & [command2 & ... ]
```

73226 The standard input for an asynchronous list, before any explicit redirections are performed, shall
 73227 be considered to be assigned to a file that has the same properties as `/dev/null`. If it is an
 73228 interactive shell, this need not happen. In all cases, explicit redirection of standard input shall
 73229 override this activity.

73230 When an element of an asynchronous list (the portion of the list ended by an `&`,
 73231 such as *command1*, above) is started by the shell, the process ID of the last command in the
 73232 asynchronous list element shall become known in the current shell execution environment; see
 73233 [Section 2.12](#) (on page 2331). This process ID shall remain known until:

- 73234 1. The command terminates and the application waits for the process ID.
- 73235 2. Another asynchronous list is invoked before "\$!" (corresponding to the previous
 73236 asynchronous list) is expanded in the current execution environment.

73237 The implementation need not retain more than the {CHILD_MAX} most recent entries in its list
 73238 of known process IDs in the current shell execution environment.

73239 Exit Status

73240 The exit status of an asynchronous list shall be zero.

73241 2.9.3.2 Sequential Lists

73242 Commands that are separated by a `<semicolon>` (`' ; '`) shall be executed sequentially.

73243 The format for executing commands sequentially shall be:

73244 *command1* [`;` *command2*] ...

73245 Each command shall be expanded and executed in the order specified.

73246 Exit Status

73247 The exit status of a sequential list shall be the exit status of the last command in the list.

73248 2.9.3.3 AND Lists

73249 The control operator `"&&"` denotes an AND list. The format shall be:

73250 *command1* [`&&` *command2*] ...

73251 First *command1* shall be executed. If its exit status is zero, *command2* shall be executed, and so on,
 73252 until a command has a non-zero exit status or there are no more commands left to execute. The
 73253 commands are expanded only if they are executed.

73254 Exit Status

73255 The exit status of an AND list shall be the exit status of the last command that is executed in the
 73256 list.

73257 2.9.3.4 OR Lists

73258 The control operator `"||"` denotes an OR List. The format shall be:

73259 *command1* [`||` *command2*] ...

73260 First, *command1* shall be executed. If its exit status is non-zero, *command2* shall be executed, and
 73261 so on, until a command has a zero exit status or there are no more commands left to execute.

Exit Status

The exit status of an OR list shall be the exit status of the last command that is executed in the list.

2.9.4 Compound Commands

The shell has several programming constructs that are “compound commands”, which provide control flow for commands. Each of these compound commands has a reserved word or control operator at the beginning, and a corresponding terminator reserved word or operator at the end. In addition, each can be followed by redirections on the same line as the terminator. Each redirection shall apply to all the commands within the compound command that do not explicitly override that redirection.

2.9.4.1 Grouping Commands

The format for grouping commands is as follows:

(*compound-list*) Execute *compound-list* in a subshell environment; see [Section 2.12](#) (on page 2331). Variable assignments and built-in commands that affect the environment shall not remain in effect after the list finishes.

{ *compound-list*; } Execute *compound-list* in the current process environment. The semicolon shown here is an example of a control operator delimiting the } reserved word. Other delimiters are possible, as shown in [Section 2.10](#) (on page 2325); a <newline> is frequently used.

Exit Status

The exit status of a grouping command shall be the exit status of *compound-list*.

2.9.4.2 The for Loop

The **for** loop shall execute a sequence of commands for each member in a list of *items*. The **for** loop requires that the reserved words **do** and **done** be used to delimit the sequence of commands.

The format for the **for** loop is as follows:

```
for name in [word ... ]
do
    compound-list
done
```

First, the list of words following **in** shall be expanded to generate a list of items. Then, the variable *name* shall be set to each item, in turn, and the *compound-list* executed each time. If no items result from the expansion, the *compound-list* shall not be executed. Omitting:

```
in word...
```

shall be equivalent to:

```
in "$@"
```

73298 **Exit Status**

73299 The exit status of a **for** command shall be the exit status of the last command that executes. If
 73300 there are no items, the exit status shall be zero.

73301 2.9.4.3 *Case Conditional Construct*

73302 The conditional construct **case** shall execute the *compound-list* corresponding to the first one of
 73303 several *patterns* (see [Section 2.13](#), on page 2332) that is matched by the string resulting from the
 73304 tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote
 73305 removal of the given word. The reserved word **in** shall denote the beginning of the patterns to
 73306 be matched. Multiple patterns with the same *compound-list* shall be delimited by the '|' symbol.
 73307 The control operator ')' terminates a list of patterns corresponding to a given action.
 73308 The *compound-list* for each list of patterns, with the possible exception of the last, shall be
 73309 terminated with ";;". The **case** construct terminates with the reserved word **esac** (**case**
 73310 reversed).

73311 The format for the **case** construct is as follows:

```
73312 case word in
73313     [(pattern1) compound-list;;
73314     [(pattern[ | pattern] ... ) compound-list;;] ...
73315     [(pattern[ | pattern] ... ) compound-list]
73316 esac
```

73317 The ";;" is optional for the last *compound-list*.

73318 In order from the beginning to the end of the **case** statement, each *pattern* that labels a *compound-*
 73319 *list* shall be subjected to tilde expansion, parameter expansion, command substitution, and
 73320 arithmetic expansion, and the result of these expansions shall be compared against the
 73321 expansion of *word*, according to the rules described in [Section 2.13](#) (on page 2332) (which also
 73322 describes the effect of quoting parts of the pattern). After the first match, no more patterns shall
 73323 be expanded, and the *compound-list* shall be executed. The order of expansion and comparison of
 73324 multiple *patterns* that label a *compound-list* statement is unspecified.

73325 **Exit Status**

73326 The exit status of **case** shall be zero if no patterns are matched. Otherwise, the exit status shall be
 73327 the exit status of the last command executed in the *compound-list*.

73328 2.9.4.4 *The if Conditional Construct*

73329 The **if** command shall execute a *compound-list* and use its exit status to determine whether to
 73330 execute another *compound-list*.

73331 The format for the **if** construct is as follows:

```
73332 if compound-list
73333 then
73334     compound-list
73335 [elif compound-list
73336 then
73337     compound-list] ...
73338 [else
73339     compound-list]
73340 fi
```

The **if** *compound-list* shall be executed; if its exit status is zero, the **then** *compound-list* shall be executed and the command shall complete. Otherwise, each **elif** *compound-list* shall be executed, in turn, and if its exit status is zero, the **then** *compound-list* shall be executed and the command shall complete. Otherwise, the **else** *compound-list* shall be executed.

Exit Status

The exit status of the **if** command shall be the exit status of the **then** or **else** *compound-list* that was executed, or zero, if none was executed.

2.9.4.5 The while Loop

The **while** loop shall continuously execute one *compound-list* as long as another *compound-list* has a zero exit status.

The format of the **while** loop is as follows:

```
while compound-list-1
do
    compound-list-2
done
```

The *compound-list-1* shall be executed, and if it has a non-zero exit status, the **while** command shall complete. Otherwise, the *compound-list-2* shall be executed, and the process shall repeat.

Exit Status

The exit status of the **while** loop shall be the exit status of the last *compound-list-2* executed, or zero if none was executed.

2.9.4.6 The until Loop

The **until** loop shall continuously execute one *compound-list* as long as another *compound-list* has a non-zero exit status.

The format of the **until** loop is as follows:

```
until compound-list-1
do
    compound-list-2
done
```

The *compound-list-1* shall be executed, and if it has a zero exit status, the **until** command completes. Otherwise, the *compound-list-2* shall be executed, and the process repeats.

Exit Status

The exit status of the **until** loop shall be the exit status of the last *compound-list-2* executed, or zero if none was executed.

2.9.5 Function Definition Command

A function is a user-defined name that is used as a simple command to call a compound command with new positional parameters. A function is defined with a “function definition command”.

The format of a function definition command is as follows:

```
fname() compound-command[io-redirect ...]
```

The function is named *fname*; the application shall ensure that it is a name (see XBD Section 3.230, on page 70). An implementation may allow other characters in a function name as an extension. The implementation shall maintain separate name spaces for functions and variables.

The argument *compound-command* represents a compound command, as described in Section 2.9.4 (on page 2321).

When the function is declared, none of the expansions in Section 2.6 (on page 2305) shall be performed on the text in *compound-command* or *io-redirect*; all expansions shall be performed as normal each time the function is called. Similarly, the optional *io-redirect* redirections and any variable assignments within *compound-command* shall be performed during the execution of the function itself, not the function definition. See Section 2.8.1 (on page 2315) for the consequences of failures of these operations on interactive and non-interactive shells.

When a function is executed, it shall have the syntax-error and variable-assignment properties described for special built-in utilities in the enumerated list at the beginning of Section 2.14 (on page 2334).

The *compound-command* shall be executed whenever the function name is specified as the name of a simple command (see Section 2.9.1.1, on page 2317). The operands to the command temporarily shall become the positional parameters during the execution of the *compound-command*; the special parameter '#' also shall be changed to reflect the number of operands. The special parameter 0 shall be unchanged. When the function completes, the values of the positional parameters and the special parameter '#' shall be restored to the values they had before the function was executed. If the special built-in *return* is executed in the *compound-command*, the function completes and execution shall resume with the next command after the function call.

Exit Status

The exit status of a function definition shall be zero if the function was declared successfully; otherwise, it shall be greater than zero. The exit status of a function invocation shall be the exit status of the last command executed by the function.

2.10 Shell Grammar

The following grammar defines the Shell Command Language. This formal syntax shall take precedence over the preceding text syntax description.

2.10.1 Shell Grammar Lexical Conventions

The input language to the shell must be first recognized at the character level. The resulting tokens shall be classified by their immediate context according to the following rules (applied in order). These rules shall be used to determine what a “token” is that is subject to parsing at the token level. The rules for token recognition in [Section 2.3](#) (on page 2299) shall apply.

1. A <newline> shall be returned as the token identifier **NEWLINE**.
2. If the token is an operator, the token identifier for that operator shall result.
3. If the string consists solely of digits and the delimiter character is one of '<' or '>', the token identifier **IO_NUMBER** shall be returned.
4. Otherwise, the token identifier **TOKEN** results.

Further distinction on **TOKEN** is context-dependent. It may be that the same **TOKEN** yields **WORD**, a **NAME**, an **ASSIGNMENT**, or one of the reserved words below, dependent upon the context. Some of the productions in the grammar below are annotated with a rule number from the following list. When a **TOKEN** is seen where one of those annotated productions could be used to reduce the symbol, the applicable rule shall be applied to convert the token identifier type of the **TOKEN** to a token identifier acceptable at that point in the grammar. The reduction shall then proceed based upon the token identifier type yielded by the rule applied. When more than one rule applies, the highest numbered rule shall apply (which in turn may refer to another rule). (Note that except in rule 7, the presence of an '=' in the token has no effect.)

The **WORD** tokens shall have the word expansion rules applied to them immediately before the associated command is executed, not at the time the command is parsed.

2.10.2 Shell Grammar Rules

1. [Command Name]

When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word shall result. Otherwise, the token **WORD** shall be returned. Also, if the parser is in any state where only a reserved word could be the next correct token, proceed as above.

Note: Because at this point <quotation-mark> characters are retained in the token, quoted strings cannot be recognized as reserved words. This rule also implies that reserved words are not recognized except in certain positions in the input, such as after a <newline> or <semicolon>; the grammar presumes that if the reserved word is intended, it is properly delimited by the user, and does not attempt to reflect that requirement directly. Also note that line joining is done before tokenization, as described in [Section 2.2.1](#) (on page 2298), so escaped <newline> characters are already removed at this point.

Rule 1 is not directly referenced in the grammar, but is referred to by other rules, or applies globally.

2. [Redirection to or from filename]

The expansions specified in [Section 2.7](#) (on page 2312) shall occur. As specified there, exactly one field can result (or the result is unspecified), and there are additional

- 73449 requirements on pathname expansion.
- 73450 3. [Redirection from here-document]
- 73451 Quote removal shall be applied to the word to determine the delimiter that is used to find
- 73452 the end of the here-document that begins after the next <newline>.
- 73453 4. [Case statement termination]
- 73454 When the **TOKEN** is exactly the reserved word **esac**, the token identifier for **esac** shall
- 73455 result. Otherwise, the token **WORD** shall be returned.
- 73456 5. [**NAME** in **for**]
- 73457 When the **TOKEN** meets the requirements for a name (see XBD [Section 3.230](#), on page
- 73458 70), the token identifier **NAME** shall result. Otherwise, the token **WORD** shall be
- 73459 returned.
- 73460 6. [Third word of **for** and **case**]
- 73461 a. [**case** only]
- 73462 When the **TOKEN** is exactly the reserved word **in**, the token identifier for **in** shall
- 73463 result. Otherwise, the token **WORD** shall be returned.
- 73464 b. [**for** only]
- 73465 When the **TOKEN** is exactly the reserved word **in** or **do**, the token identifier for **in**
- 73466 or **do** shall result, respectively. Otherwise, the token **WORD** shall be returned.
- 73467 (For a. and b.: As indicated in the grammar, a *linebreak* precedes the tokens **in** and **do**. If
- 73468 <newline> characters are present at the indicated location, it is the token after them that is
- 73469 treated in this fashion.)
- 73470 7. [Assignment preceding command name]
- 73471 a. [When the first word]
- 73472 If the **TOKEN** does not contain the character '=' , rule 1 is applied. Otherwise, 7b
- 73473 shall be applied.
- 73474 b. [Not the first word]
- 73475 If the **TOKEN** contains the <equals-sign> character:
- 73476 — If it begins with '=' , the token **WORD** shall be returned.
- 73477 — If all the characters preceding '=' form a valid name (see XBD [Section 3.230](#),
- 73478 on page 70), the token **ASSIGNMENT_WORD** shall be returned. (Quoted
- 73479 characters cannot participate in forming a valid name.)
- 73480 — Otherwise, it is unspecified whether it is **ASSIGNMENT_WORD** or **WORD**
- 73481 that is returned.
- 73482 Assignment to the **NAME** shall occur as specified in [Section 2.9.1](#) (on page 2316).
- 73483 8. [**NAME** in function]
- 73484 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word
- 73485 shall result. Otherwise, when the **TOKEN** meets the requirements for a name, the token
- 73486 identifier **NAME** shall result. Otherwise, rule 7 applies.
- 73487 9. [Body of function]
- 73488 Word expansion and assignment shall never occur, even when required by the rules

above, when this rule is being parsed. Each **TOKEN** that might either be expanded or have assignment applied to it shall instead be returned as a single **WORD** consisting only of characters that are exactly the token described in [Section 2.3](#) (on page 2299).

```

73492  /* -----
73493  The grammar symbols
73494  ----- */
73495  %token  WORD
73496  %token  ASSIGNMENT_WORD
73497  %token  NAME
73498  %token  NEWLINE
73499  %token  IO_NUMBER

73500  /* The following are the operators mentioned above. */
73501  %token  AND_IF      OR_IF      DSEMI
73502  /*      '&&'      '||'      ';;'      */
73503  %token  DLESS      DGREAT      LESSAND      GREATAND      LESSGREAT      DLESSDASH
73504  /*      '<<'      '>>'      '<&'      '>&'      '<>'      '<<-'      */
73505  %token  CLOBBER
73506  /*      '>|'      */
73507  /* The following are the reserved words. */
73508  %token  If      Then      Else      Elif      Fi      Do      Done
73509  /*      'if'      'then'      'else'      'elif'      'fi'      'do'      'done'      */
73510  %token  Case      Esac      While      Until      For
73511  /*      'case'      'esac'      'while'      'until'      'for'      */
73512  /* These are reserved words, not operator tokens, and are
73513  recognized when reserved words are recognized. */
73514  %token  Lbrace      Rbrace      Bang
73515  /*      '{'      '}'      '!'      */
73516  %token  In
73517  /*      'in'      */
73518  /* -----
73519  The Grammar
73520  ----- */
73521  %start  complete_command
73522  %%
73523  complete_command : list separator
73524                  | list
73525                  ;
73526  list             : list separator_op and_or
73527                  |
73528                  and_or
73529                  ;
73529  and_or           :
73530                  | and_or AND_IF linebreak pipeline
73531                  | and_or OR_IF linebreak pipeline
73532                  ;
73533  pipeline         : pipe_sequence
73534                  | Bang pipe_sequence
73535                  ;

```

```

73536     pipe_sequence      :                               command
73537                       | pipe_sequence '|' linebreak command
73538                       ;
73539     command              : simple_command
73540                       | compound_command
73541                       | compound_command redirect_list
73542                       | function_definition
73543                       ;
73544     compound_command      : brace_group
73545                       | subshell
73546                       | for_clause
73547                       | case_clause
73548                       | if_clause
73549                       | while_clause
73550                       | until_clause
73551                       ;
73552     subshell              : '(' compound_list ')'
73553                       ;
73554     compound_list         :                               term
73555                       | newline_list term
73556                       |                               term separator
73557                       | newline_list term separator
73558                       ;
73559     term                  : term separator and_or
73560                       |                               and_or
73561                       ;
73562     for_clause            : For name linebreak                               do_group
73563                       | For name linebreak in                               sequential_sep do_group
73564                       | For name linebreak in wordlist sequential_sep do_group
73565                       ;
73566     name                  : NAME /* Apply rule 5 */
73567                       ;
73568     in                    : In /* Apply rule 6 */
73569                       ;
73570     wordlist              : wordlist WORD
73571                       |                               WORD
73572                       ;
73573     case_clause           : Case WORD linebreak in linebreak case_list      Esac
73574                       | Case WORD linebreak in linebreak case_list_ns Esac
73575                       | Case WORD linebreak in linebreak                               Esac
73576                       ;
73577     case_list_ns          : case_list case_item_ns
73578                       |                               case_item_ns
73579                       ;
73580     case_list             : case_list case_item
73581                       |                               case_item
73582                       ;
73583     case_item_ns          : pattern ')' linebreak
73584                       | pattern ')' compound_list linebreak
73585                       | '(' pattern ')' linebreak
73586                       | '(' pattern ')' compound_list linebreak
73587                       ;
73588     case_item             : pattern ')' linebreak DSEMI linebreak

```

Shell Command Language

Shell Grammar

```

73589         |      pattern ')' compound_list DSEMI linebreak
73590         | ' (' pattern ')' linebreak      DSEMI linebreak
73591         | ' (' pattern ')' compound_list DSEMI linebreak
73592         ;
73593 pattern      :      WORD      /* Apply rule 4 */
73594         | pattern '|' WORD      /* Do not apply rule 4 */
73595         ;
73596 if_clause    : If compound_list Then compound_list else_part Fi
73597         | If compound_list Then compound_list      Fi
73598         ;
73599 else_part    : Elif compound_list Then else_part
73600         | Else compound_list
73601         ;
73602 while_clause : While compound_list do_group
73603         ;
73604 until_clause : Until compound_list do_group
73605         ;
73606 function_definition : fname '(' ')' linebreak function_body
73607         ;
73608 function_body  : compound_command      /* Apply rule 9 */
73609         | compound_command redirect_list /* Apply rule 9 */
73610         ;
73611 fname          : NAME      /* Apply rule 8 */
73612         ;
73613 brace_group    : Lbrace compound_list Rbrace
73614         ;
73615 do_group       : Do compound_list Done      /* Apply rule 6 */
73616         ;
73617 simple_command : cmd_prefix cmd_word cmd_suffix
73618         | cmd_prefix cmd_word
73619         | cmd_prefix
73620         | cmd_name cmd_suffix
73621         | cmd_name
73622         ;
73623 cmd_name       : WORD      /* Apply rule 7a */
73624         ;
73625 cmd_word       : WORD      /* Apply rule 7b */
73626         ;
73627 cmd_prefix     :      io_redirect
73628         | cmd_prefix io_redirect
73629         |      ASSIGNMENT_WORD
73630         | cmd_prefix ASSIGNMENT_WORD
73631         ;
73632 cmd_suffix     :      io_redirect
73633         | cmd_suffix io_redirect
73634         |      WORD
73635         | cmd_suffix WORD
73636         ;
73637 redirect_list  :      io_redirect
73638         | redirect_list io_redirect
73639         ;
73640 io_redirect    :      io_file
73641         | IO_NUMBER io_file

```

```

73642         |             io_here
73643         | IO_NUMBER io_here
73644         ;
73645     io_file      : '<'         filename
73646                 | LESSAND    filename
73647                 | '>'         filename
73648                 | GREATAND   filename
73649                 | DGREAT     filename
73650                 | LESSGREAT  filename
73651                 | CLOBBER    filename
73652         ;
73653     filename      : WORD                      /* Apply rule 2 */
73654         ;
73655     io_here       : DLESS      here_end
73656                 | DLESSDASH here_end
73657         ;
73658     here_end      : WORD                      /* Apply rule 3 */
73659         ;
73660     newline_list  : NEWLINE
73661                 | newline_list NEWLINE
73662         ;
73663     linebreak     : newline_list
73664                 | /* empty */
73665         ;
73666     separator_op  : '&'
73667                 | ';'
73668         ;
73669     separator     : separator_op linebreak
73670                 | newline_list
73671         ;
73672     sequential_sep : ';' linebreak
73673                 | newline_list
73674         ;

```

2.11 Signals and Error Handling

When a command is in an asynchronous list, it shall inherit from the shell a signal action of ignored (SIG_IGN) for the SIGQUIT and SIGINT signals, and may inherit a signal mask in which SIGQUIT and SIGINT are blocked. Otherwise, the signal actions and signal mask inherited by the command shall be the same as those inherited by the shell from its parent unless a signal action is modified by the *trap* special built-in (see *trap*)

When a signal for which a trap has been set is received while the shell is waiting for the completion of a utility executing a foreground command, the trap associated with that signal shall not be executed until after the foreground command has completed. When the shell is waiting, by means of the *wait* utility, for asynchronous commands to complete, the reception of a signal for which a trap has been set shall cause the *wait* utility to return immediately with an exit status >128, immediately after which the trap associated with that signal shall be taken.

If multiple signals are pending for the shell for which there are associated trap actions, the order of execution of trap actions is unspecified.

2.12 Shell Execution Environment

A shell execution environment consists of the following:

- Open files inherited upon invocation of the shell, plus open files controlled by *exec*
- Working directory as set by *cd*
- File creation mask set by *umask*
- Current traps set by *trap*
- Shell parameters that are set by variable assignment (see the *set* special built-in) or from the System Interfaces volume of POSIX.1-2008 environment inherited by the shell when it begins (see the *export* special built-in)
- Shell functions; see [Section 2.9.5](#) (on page 2324)
- Options turned on at invocation or by *set*
- Process IDs of the last commands in asynchronous lists known to this shell environment; see [Section 2.9.3.1](#) (on page 2319)
- Shell aliases; see [Section 2.3.1](#) (on page 2300)

Utilities other than the special built-ins (see [Section 2.14](#), on page 2334) shall be invoked in a separate environment that consists of the following. The initial value of these objects shall be the same as that for the parent shell, except as noted below.

- Open files inherited on invocation of the shell, open files controlled by the *exec* special built-in plus any modifications, and additions specified by any redirections to the utility
- Current working directory
- File creation mask
- If the utility is a shell script, traps caught by the shell shall be set to the default values and traps ignored by the shell shall be set to be ignored by the utility; if the utility is not a shell script, the trap actions (default or ignore) shall be mapped into the appropriate signal handling actions for the utility
- Variables with the *export* attribute, along with those explicitly exported for the duration of the command, shall be passed to the utility environment variables

The environment of the shell process shall not be changed by the utility unless explicitly specified by the utility description (for example, *cd* and *umask*).

A subshell environment shall be created as a duplicate of the shell environment, except that signal traps set by that shell environment shall be set to the default values. Changes made to the subshell environment shall not affect the shell environment. Command substitution, commands that are grouped with parentheses, and asynchronous lists shall be executed in a subshell environment. Additionally, each command of a multi-command pipeline is in a subshell environment; as an extension, however, any or all commands in a pipeline may be executed in the current environment. All other commands shall be executed in the current shell environment.

2.13 Pattern Matching Notation

The pattern matching notation described in this section is used to specify patterns for matching strings in the shell. Historically, pattern matching notation is related to, but slightly different from, the regular expression notation described in XBD [Chapter 9](#) (on page 181). For this reason, the description of the rules for this pattern matching notation are based on the description of regular expression notation, modified to account for the differences.

2.13.1 Patterns Matching a Single Character

The following patterns matching a single character shall match a single character: ordinary characters, special pattern characters, and pattern bracket expressions. The pattern bracket expression also shall match a single collating element. A `<backslash>` character shall escape the following character. The escaping `<backslash>` shall be discarded.

An ordinary character is a pattern that shall match itself. It can be any character in the supported character set except for NUL, those special shell characters in [Section 2.2](#) (on page 2298) that require quoting, and the following three special pattern characters. Matching shall be based on the bit pattern used for encoding the character, not on the graphic representation of the character. If any character (ordinary, shell special, or pattern special) is quoted, that pattern shall match the character itself. The shell special characters always require quoting.

When unquoted and outside a bracket expression, the following three characters shall have special meaning in the specification of patterns:

- ? A `<question-mark>` is a pattern that shall match any character.
- * An `<asterisk>` is a pattern that shall match multiple characters, as described in [Section 2.13.2](#).
- [If an open bracket introduces a bracket expression as in XBD [Section 9.3.5](#) (on page 184), except that the `<exclamation-mark>` character (`'!'`) shall replace the `<circumflex>` character (`'^'`) in its role in a non-matching list in the regular expression notation, it shall introduce a pattern bracket expression. A bracket expression starting with an unquoted `<circumflex>` character produces unspecified results. Otherwise, `'['` shall match the character itself.

When pattern matching is used where shell quote removal is not performed (such as in the argument to the `find -name` primary when `find` is being called using one of the `exec` functions as defined in the System Interfaces volume of POSIX.1-2008, or in the `pattern` argument to the `fnmatch()` function), special characters can be escaped to remove their special meaning by preceding them with a `<backslash>` character. This escaping `<backslash>` is discarded. The sequence `"\\\"` represents one literal `<backslash>`. All of the requirements and effects of quoting on ordinary, shell special, and special pattern characters shall apply to escaping in this context.

2.13.2 Patterns Matching Multiple Characters

The following rules are used to construct patterns matching multiple characters from patterns matching a single character:

1. The `<asterisk>` (`'*'`) is a pattern that shall match any string, including the null string.
2. The concatenation of patterns matching a single character is a valid pattern that shall match the concatenation of the single characters or collating elements matched by each of the concatenated patterns.

3. The concatenation of one or more patterns matching a single character with one or more <asterisk> characters is a valid pattern. In such patterns, each <asterisk> shall match a string of zero or more characters, matching the greatest possible number of characters that still allows the remainder of the pattern to match the string.

2.13.3 Patterns Used for Filename Expansion

The rules described so far in [Section 2.13.1](#) (on page 2332) and [Section 2.13.2](#) (on page 2332) are qualified by the following rules that apply when pattern matching notation is used for filename expansion:

1. The <slash> character in a pathname shall be explicitly matched by using one or more <slash> characters in the pattern; it shall neither be matched by the <asterisk> or <question-mark> special characters nor by a bracket expression. <slash> characters in the pattern shall be identified before bracket expressions; thus, a <slash> cannot be included in a pattern bracket expression used for filename expansion. If a <slash> character is found following an unescaped <left-square-bracket> character before a corresponding <right-square-bracket> is found, the open bracket shall be treated as an ordinary character. For example, the pattern "a[b/c]d" does not match such pathnames as **abd** or **a/d**. It only matches a pathname of literally **a[b/c]d**.
2. If a filename begins with a <period> ('.'), the <period> shall be explicitly matched by using a <period> as the first character of the pattern or immediately following a <slash> character. The leading <period> shall not be matched by:
 - The <asterisk> or <question-mark> special characters
 - A bracket expression containing a non-matching list, such as "[!a]", a range expression, such as "[%~0]", or a character class expression, such as "[[:punct:]]"

It is unspecified whether an explicit <period> in a bracket expression matching list, such as "[.abc]", can match a leading <period> in a filename.

3. Specified patterns shall be matched against existing filenames and pathnames, as appropriate. Each component that contains a pattern character shall require read permission in the directory containing that component. Any component, except the last, that does not contain a pattern character shall require search permission. For example, given the pattern:

```
/foo/bar/x*/bam
```

search permission is needed for directories **/** and **foo**, search and read permissions are needed for directory **bar**, and search permission is needed for each **x*** directory. If the pattern matches any existing filenames or pathnames, the pattern shall be replaced with those filenames and pathnames, sorted according to the collating sequence in effect in the current locale.

If the pattern contains an open bracket ('[') that does not introduce a bracket expression as in XBD [Section 9.3.5](#) (on page 184), it is unspecified whether other unquoted pattern matching characters within the same slash-delimited component of the pattern retain their special meanings or are treated as ordinary characters. For example, the pattern "a*[/b*" may match all filenames beginning with 'b' in the directory "a*[" or it may match all filenames beginning with 'b' in all directories with names beginning with 'a' and ending with '['.

If the pattern does not match any existing filenames or pathnames, the pattern string shall

73813 be left unchanged.

73814 2.14 Special Built-In Utilities

73815 The following “special built-in” utilities shall be supported in the shell command language. The
73816 output of each command, if any, shall be written to standard output, subject to the normal
73817 redirection and piping possible with all commands.

73818 The term “built-in” implies that the shell can execute the utility directly and does not need to
73819 search for it. An implementation may choose to make any utility a built-in; however, the special
73820 built-in utilities described here differ from regular built-in utilities in two respects:

- 73821 1. A syntax error in a special built-in utility may cause a shell executing that utility to abort,
73822 while a syntax error in a regular built-in utility shall not cause a shell executing that
73823 utility to abort. (See [Section 2.8.1](#) (on page 2315) for the consequences of errors on
73824 interactive and non-interactive shells.) If a special built-in utility encountering a syntax
73825 error does not abort the shell, its exit value shall be non-zero.
- 73826 2. Variable assignments specified with special built-in utilities remain in effect after the
73827 built-in completes; this shall not be the case with a regular built-in or other utility.

73828 The special built-in utilities in this section need not be provided in a manner accessible via the
73829 *exec* family of functions defined in the System Interfaces volume of POSIX.1-2008.

73830 Some of the special built-ins are described as conforming to XBD [Section 12.2](#) (on page 215). For
73831 those that are not, the requirement in [Section 14](#) (on page 2288) that “--” be recognized as a
73832 first argument to be discarded does not apply and a conforming application shall not use that
73833 argument.

*Shell Command Language***break**73834 **NAME**

73835 break — exit from for, while, or until loop

73836 **SYNOPSIS**73837 break [*n*]73838 **DESCRIPTION**

73839 The *break* utility shall exit from the smallest enclosing **for**, **while**, or **until** loop, if any; or from
 73840 the *n*th enclosing loop if *n* is specified. The value of *n* is an unsigned decimal integer greater
 73841 than or equal to 1. The default shall be equivalent to *n*=1. If *n* is greater than the number of
 73842 enclosing loops, the outermost enclosing loop shall be exited. Execution shall continue with the
 73843 command immediately following the loop.

73844 **OPTIONS**

73845 None.

73846 **OPERANDS**

73847 See the DESCRIPTION.

73848 **STDIN**

73849 Not used.

73850 **INPUT FILES**

73851 None.

73852 **ENVIRONMENT VARIABLES**

73853 None.

73854 **ASYNCHRONOUS EVENTS**

73855 Default.

73856 **STDOUT**

73857 Not used.

73858 **STDERR**

73859 The standard error shall be used only for diagnostic messages.

73860 **OUTPUT FILES**

73861 None.

73862 **EXTENDED DESCRIPTION**

73863 None.

73864 **EXIT STATUS**

73865 0 Successful completion.

73866 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.73867 **CONSEQUENCES OF ERRORS**

73868 Default.

break*Shell Command Language*73869 **APPLICATION USAGE**

73870 None.

73871 **EXAMPLES**

```

73872     for i in *
73873     do
73874         if test -d "$i"
73875         then break
73876         fi
73877     done

```

73878 **RATIONALE**

73879 In early proposals, consideration was given to expanding the syntax of *break* and *continue* to refer
 73880 to a label associated with the appropriate loop as a preferable alternative to the *n* method.
 73881 However, this volume of POSIX.1-2008 does reserve the name space of command names ending
 73882 with a <colon>. It is anticipated that a future implementation could take advantage of this and
 73883 provide something like:

```

73884 outofloop: for i in a b c d e
73885 do
73886     for j in 0 1 2 3 4 5 6 7 8 9
73887     do
73888         if test -r "${i}${j}"
73889         then break outofloop
73890         fi
73891     done
73892 done

```

73893 and that this might be standardized after implementation experience is achieved.

73894 **FUTURE DIRECTIONS**

73895 None.

73896 **SEE ALSO**73897 [Section 2.14](#) (on page 2334)73898 **CHANGE HISTORY**73899 **Issue 6**

73900 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
 73901 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
 73902 behavior is intended.

*Shell Command Language***colon**73903 **NAME**

73904 colon — null utility

73905 **SYNOPSIS**73906 : [*argument*...]73907 **DESCRIPTION**

73908 This utility shall only expand command *arguments*. It is used when a command is needed, as in
 73909 the **then** condition of an **if** command, but nothing is to be done by the command.

73910 **OPTIONS**

73911 None.

73912 **OPERANDS**

73913 See the DESCRIPTION.

73914 **STDIN**

73915 Not used.

73916 **INPUT FILES**

73917 None.

73918 **ENVIRONMENT VARIABLES**

73919 None.

73920 **ASYNCHRONOUS EVENTS**

73921 Default.

73922 **STDOUT**

73923 Not used.

73924 **STDERR**

73925 The standard error shall be used only for diagnostic messages.

73926 **OUTPUT FILES**

73927 None.

73928 **EXTENDED DESCRIPTION**

73929 None.

73930 **EXIT STATUS**

73931 Zero.

73932 **CONSEQUENCES OF ERRORS**

73933 Default.

73934 **APPLICATION USAGE**

73935 None.

73936 **EXAMPLES**

```
73937 : ${X=abc}
73938 if     false
73939 then  :
73940 else  echo $X
73941 fi
73942 abc
```

73943 As with any of the special built-ins, the null utility can also have variable assignments and
 73944 redirections associated with it, such as:

73945 `x=y : > z`

colon*Shell Command Language*

73946 which sets variable *x* to the value *y* (so that it persists after the null utility completes) and creates
73947 or truncates file *z*.

RATIONALE

73948 None.
73949

FUTURE DIRECTIONS

73950 None.
73951

SEE ALSO

73952 [Section 2.14](#) (on page 2334)
73953

CHANGE HISTORY**Issue 6**

73955 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
73956 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
73957 behavior is intended.
73958

Issue 7

73959 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
73960

Shell Command Language

continue73961 **NAME**

73962 continue — continue for, while, or until loop

73963 **SYNOPSIS**73964 continue [*n*]73965 **DESCRIPTION**

73966 The *continue* utility shall return to the top of the smallest enclosing **for**, **while**, or **until** loop, or to
 73967 the top of the *n*th enclosing loop, if *n* is specified. This involves repeating the condition list of a
 73968 **while** or **until** loop or performing the next assignment of a **for** loop, and re-executing the loop if
 73969 appropriate.

73970 The value of *n* is a decimal integer greater than or equal to 1. The default shall be equivalent to
 73971 *n*=1. If *n* is greater than the number of enclosing loops, the outermost enclosing loop shall be
 73972 used.

73973 **OPTIONS**

73974 None.

73975 **OPERANDS**

73976 See the DESCRIPTION.

73977 **STDIN**

73978 Not used.

73979 **INPUT FILES**

73980 None.

73981 **ENVIRONMENT VARIABLES**

73982 None.

73983 **ASYNCHRONOUS EVENTS**

73984 Default.

73985 **STDOUT**

73986 Not used.

73987 **STDERR**

73988 The standard error shall be used only for diagnostic messages.

73989 **OUTPUT FILES**

73990 None.

73991 **EXTENDED DESCRIPTION**

73992 None.

73993 **EXIT STATUS**

73994 0 Successful completion.

73995 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.73996 **CONSEQUENCES OF ERRORS**

73997 Default.

continue*Shell Command Language*73998 **APPLICATION USAGE**

73999 None.

74000 **EXAMPLES**

```
74001     for i in *
74002     do
74003         if test -d "$i"
74004         then continue
74005         fi
74006         printf '%s' is not a directory.\n' "$i"
74007     done
```

74008 **RATIONALE**

74009 None.

74010 **FUTURE DIRECTIONS**

74011 None.

74012 **SEE ALSO**74013 [Section 2.14](#) (on page 2334)74014 **CHANGE HISTORY**74015 **Issue 6**

74016 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
74017 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
74018 behavior is intended.

74019 **Issue 7**74020 The example is changed to use the *printf* utility rather than *echo*.

*Shell Command Language***dot**74021 **NAME**

74022 dot — execute commands in the current environment

74023 **SYNOPSIS**74024 . *file*74025 **DESCRIPTION**74026 The shell shall execute commands from the *file* in the current environment.

74027 If *file* does not contain a <slash>, the shell shall use the search path specified by *PATH* to find the
 74028 directory containing *file*. Unlike normal command search, however, the file searched for by the
 74029 *dot* utility need not be executable. If no readable file is found, a non-interactive shell shall abort;
 74030 an interactive shell shall write a diagnostic message to standard error, but this condition shall
 74031 not be considered a syntax error.

74032 **OPTIONS**

74033 None.

74034 **OPERANDS**

74035 See the DESCRIPTION.

74036 **STDIN**

74037 Not used.

74038 **INPUT FILES**

74039 See the DESCRIPTION.

74040 **ENVIRONMENT VARIABLES**

74041 See the DESCRIPTION.

74042 **ASYNCHRONOUS EVENTS**

74043 Default.

74044 **STDOUT**

74045 Not used.

74046 **STDERR**

74047 The standard error shall be used only for diagnostic messages.

74048 **OUTPUT FILES**

74049 None.

74050 **EXTENDED DESCRIPTION**

74051 None.

74052 **EXIT STATUS**

74053 Returns the value of the last command executed, or a zero exit status if no command is executed.

74054 **CONSEQUENCES OF ERRORS**

74055 Default.

dot*Shell Command Language*74056 **APPLICATION USAGE**

74057 None.

74058 **EXAMPLES**

```
74059 cat foobar
74060 foo=hello bar=world
74061 . ./foobar
74062 echo $foo $bar
74063 hello world
```

74064 **RATIONALE**

74065 Some older implementations searched the current directory for the *file*, even if the value of *PATH*
74066 disallowed it. This behavior was omitted from this volume of POSIX.1-2008 due to concerns
74067 about introducing the susceptibility to trojan horses that the user might be trying to avoid by
74068 leaving **dot** out of *PATH*.

74069 The KornShell version of *dot* takes optional arguments that are set to the positional parameters.
74070 This is a valid extension that allows a *dot* script to behave identically to a function.

74071 **FUTURE DIRECTIONS**

74072 None.

74073 **SEE ALSO**74074 [Section 2.14](#) (on page 2334)74075 **CHANGE HISTORY**74076 **Issue 6**

74077 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
74078 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
74079 behavior is intended.

74080 **Issue 7**

74081 SD5-XCU-ERN-164 is applied.

*Shell Command Language***eval**74082 **NAME**

74083 eval — construct command by concatenating arguments

74084 **SYNOPSIS**74085 eval [*argument*...]74086 **DESCRIPTION**74087 The *eval* utility shall construct a command by concatenating *arguments* together, separating each
74088 with a <space> character. The constructed command shall be read and executed by the shell.74089 **OPTIONS**

74090 None.

74091 **OPERANDS**

74092 See the DESCRIPTION.

74093 **STDIN**

74094 Not used.

74095 **INPUT FILES**

74096 None.

74097 **ENVIRONMENT VARIABLES**

74098 None.

74099 **ASYNCHRONOUS EVENTS**

74100 Default.

74101 **STDOUT**

74102 Not used.

74103 **STDERR**

74104 The standard error shall be used only for diagnostic messages.

74105 **OUTPUT FILES**

74106 None.

74107 **EXTENDED DESCRIPTION**

74108 None.

74109 **EXIT STATUS**74110 If there are no *arguments*, or only null arguments, *eval* shall return a zero exit status; otherwise, it
74111 shall return the exit status of the command defined by the string of concatenated *arguments*
74112 separated by <space> characters.74113 **CONSEQUENCES OF ERRORS**

74114 Default.

74115 **APPLICATION USAGE**

74116 None.

74117 **EXAMPLES**

74118 foo=10 x=foo

74119 y=' \$ ' \$x

74120 echo \$y

74121 **\$foo**

74122 eval y=' \$ ' \$x

74123 echo \$y

74124 **10**

eval*Shell Command Language*74125 **RATIONALE**

74126 None.

74127 **FUTURE DIRECTIONS**

74128 None.

74129 **SEE ALSO**74130 [Section 2.14](#) (on page 2334)74131 **CHANGE HISTORY**74132 **Issue 6**

74133 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
74134 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
74135 behavior is intended.

74136 **Issue 7**

74137 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

74138 **NAME**74139 `exec` — execute commands and open, close, or copy file descriptors74140 **SYNOPSIS**74141 `exec` [*command* [*argument*...]]74142 **DESCRIPTION**74143 The *exec* utility shall open, close, and/or copy file descriptors as specified by any redirections as
74144 part of the command.74145 If *exec* is specified without *command* or *arguments*, and any file descriptors with numbers greater
74146 than 2 are opened with associated redirection statements, it is unspecified whether those file
74147 descriptors remain open when the shell invokes another utility. Scripts concerned that child
74148 shells could misuse open file descriptors can always close them explicitly, as shown in one of the
74149 following examples.74150 If *exec* is specified with *command*, it shall replace the shell with *command* without creating a new
74151 process. If *arguments* are specified, they shall be arguments to *command*. Redirection affects the
74152 current shell execution environment.74153 **OPTIONS**

74154 None.

74155 **OPERANDS**

74156 See the DESCRIPTION.

74157 **STDIN**

74158 Not used.

74159 **INPUT FILES**

74160 None.

74161 **ENVIRONMENT VARIABLES**

74162 None.

74163 **ASYNCHRONOUS EVENTS**

74164 Default.

74165 **STDOUT**

74166 Not used.

74167 **STDERR**

74168 The standard error shall be used only for diagnostic messages.

74169 **OUTPUT FILES**

74170 None.

74171 **EXTENDED DESCRIPTION**

74172 None.

74173 **EXIT STATUS**74174 If *command* is specified, *exec* shall not return to the shell; rather, the exit status of the process shall
74175 be the exit status of the program implementing *command*, which overlaid the shell. If *command* is
74176 not found, the exit status shall be 127. If *command* is found, but it is not an executable utility, the
74177 exit status shall be 126. If a redirection error occurs (see [Section 2.8.1](#), on page 2315), the shell
74178 shall exit with a value in the range 1–125. Otherwise, *exec* shall return a zero exit status.

exec*Shell Command Language*74179 **CONSEQUENCES OF ERRORS**

74180 Default.

74181 **APPLICATION USAGE**

74182 None.

74183 **EXAMPLES**74184 Open *readfile* as file descriptor 3 for reading:74185 `exec 3< readfile`74186 Open *writefile* as file descriptor 4 for writing:74187 `exec 4> writefile`

74188 Make file descriptor 5 a copy of file descriptor 0:

74189 `exec 5<&0`

74190 Close file descriptor 3:

74191 `exec 3<&-`74192 Cat the file **maggie** by replacing the current shell with the *cat* utility:74193 `exec cat maggie`74194 **RATIONALE**

74195 Most historical implementations were not conformant in that:

74196 `foo=bar exec cmd`74197 did not pass **foo** to **cmd**.74198 **FUTURE DIRECTIONS**

74199 None.

74200 **SEE ALSO**74201 [Section 2.14](#) (on page 2334)74202 **CHANGE HISTORY**74203 **Issue 6**

74204 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
 74205 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
 74206 behavior is intended.

74207 **Issue 7**

74208 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Shell Command Language

exit74209 **NAME**74210 `exit` — cause the shell to exit74211 **SYNOPSIS**74212 `exit` [*n*]74213 **DESCRIPTION**

74214 The *exit* utility shall cause the shell to exit with the exit status specified by the unsigned decimal
 74215 integer *n*. If *n* is specified, but its value is not between 0 and 255 inclusively, the exit status is
 74216 undefined.

74217 A *trap* on **EXIT** shall be executed before the shell terminates, except when the *exit* utility is
 74218 invoked in that *trap* itself, in which case the shell shall exit immediately.

74219 **OPTIONS**

74220 None.

74221 **OPERANDS**

74222 See the DESCRIPTION.

74223 **STDIN**

74224 Not used.

74225 **INPUT FILES**

74226 None.

74227 **ENVIRONMENT VARIABLES**

74228 None.

74229 **ASYNCHRONOUS EVENTS**

74230 Default.

74231 **STDOUT**

74232 Not used.

74233 **STDERR**

74234 The standard error shall be used only for diagnostic messages.

74235 **OUTPUT FILES**

74236 None.

74237 **EXTENDED DESCRIPTION**

74238 None.

74239 **EXIT STATUS**

74240 The exit status shall be *n*, if specified. Otherwise, the value shall be the exit value of the last
 74241 command executed, or zero if no command was executed. When *exit* is executed in a *trap* action,
 74242 the last command is considered to be the command that executed immediately preceding the
 74243 *trap* action.

74244 **CONSEQUENCES OF ERRORS**

74245 Default.

exit*Shell Command Language*74246 **APPLICATION USAGE**

74247 None.

74248 **EXAMPLES**74249 Exit with a *true* value:74250 `exit 0`74251 Exit with a *false* value:74252 `exit 1`74253 **RATIONALE**74254 As explained in other sections, certain exit status values have been reserved for special uses and
74255 should be used by applications only for those purposes:

74256 126 A file to be executed was found, but it was not an executable utility.

74257 127 A utility to be executed was not found.

74258 >128 A command was interrupted by a signal.

74259 **FUTURE DIRECTIONS**

74260 None.

74261 **SEE ALSO**74262 [Section 2.14](#) (on page 2334)74263 **CHANGE HISTORY**74264 **Issue 6**74265 IEEE Std 1003.1-2001/Cor 1-2002, item XCB/TC1/D6/5 is applied so that the reference page
74266 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
74267 behavior is intended.

NAME

export — set the export attribute for variables

SYNOPSIS

export *name* [=*word*] . . .

export -p

DESCRIPTION

The shell shall give the *export* attribute to the variables corresponding to the specified *names*, which shall cause them to be in the environment of subsequently executed commands. If the name of a variable is followed by =*word*, then the value of that variable shall be set to *word*.

The *export* special built-in shall support XBD [Section 12.2](#) (on page 215).

When -p is specified, *export* shall write to the standard output the names and values of all exported variables, in the following format:

"export %s=%s\n", <name>, <value>

if *name* is set, and:

"export %s\n", <name>

if *name* is unset.

The shell shall format the output, including the proper use of quoting, so that it is suitable for reinput to the shell as commands that achieve the same exporting results, except:

1. Read-only variables with values cannot be reset.
2. Variables that were unset at the time they were output need not be reset to the unset state if a value is assigned to the variable between the time the state was saved and the time at which the saved output is reinput to the shell.

When no arguments are given, the results are unspecified.

OPTIONS

See the DESCRIPTION.

OPERANDS

See the DESCRIPTION.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See the DESCRIPTION.

STDERR

The standard error shall be used only for diagnostic messages.

export*Shell Command Language*74307 **OUTPUT FILES**

74308 None.

74309 **EXTENDED DESCRIPTION**

74310 None.

74311 **EXIT STATUS**

74312 Zero.

74313 **CONSEQUENCES OF ERRORS**

74314 Default.

74315 **APPLICATION USAGE**

74316 None.

74317 **EXAMPLES**74318 Export *PWD* and *HOME* variables:74319 `export PWD HOME`74320 Set and export the *PATH* variable:74321 `export PATH=/local/bin:$PATH`

74322 Save and restore all exported variables:

74323 `export -p > temp-file`74324 `unset a lot of variables`74325 `... processing`74326 `. temp-file`74327 **RATIONALE**

74328 Some historical shells use the no-argument case as the functional equivalent of what is required
 74329 here with **-p**. This feature was left unspecified because it is not historical practice in all shells,
 74330 and some scripts may rely on the now-unspecified results on their implementations. Attempts to
 74331 specify the **-p** output as the default case were unsuccessful in achieving consensus. The **-p**
 74332 option was added to allow portable access to the values that can be saved and then later restored
 74333 using; for example, a *dot* script.

74334 **FUTURE DIRECTIONS**

74335 None.

74336 **SEE ALSO**74337 [Section 2.14](#) (on page 2334)74338 [XBD Section 12.2](#) (on page 215)74339 **CHANGE HISTORY**74340 **Issue 6**

74341 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

74342 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
 74343 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
 74344 behavior is intended.

74345 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/6 is applied, adding the following text to
 74346 the end of the first paragraph of the DESCRIPTION: "If the name of a variable is followed by
 74347 *=word*, then the value of that variable shall be set to *word*.". The reason for this change is that the
 74348 SYNOPSIS for *export* includes:

74349 `export name [=word] . . .`

74350 but the meaning of the optional “=word” is never explained in the text.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

readonly

Shell Command Language

74351 **NAME**

74352 readonly — set the readonly attribute for variables

74353 **SYNOPSIS**

74354 readonly name [=word] . . .

74355 readonly -p

74356 **DESCRIPTION**

74357 The variables whose *names* are specified shall be given the *readonly* attribute. The values of
 74358 variables with the *readonly* attribute cannot be changed by subsequent assignment, nor can those
 74359 variables be unset by the *unset* utility. If the name of a variable is followed by =*word*, then the
 74360 value of that variable shall be set to *word*.

74361 The *readonly* special built-in shall support XBD [Section 12.2](#) (on page 215).

74362 When **-p** is specified, *readonly* writes to the standard output the names and values of all read-
 74363 only variables, in the following format:

74364 "readonly %s=%s\n", <name>, <value>

74365 if *name* is set, and

74366 "readonly %s\n", <name>

74367 if *name* is unset.

74368 The shell shall format the output, including the proper use of quoting, so that it is suitable for
 74369 reinput to the shell as commands that achieve the same value and *readonly* attribute-setting
 74370 results in a shell execution environment in which:

- 74371 1. Variables with values at the time they were output do not have the *readonly* attribute set.
- 74372 2. Variables that were unset at the time they were output do not have a value at the time at
 74373 which the saved output is reinput to the shell.

74374 When no arguments are given, the results are unspecified.

74375 **OPTIONS**

74376 See the DESCRIPTION.

74377 **OPERANDS**

74378 See the DESCRIPTION.

74379 **STDIN**

74380 Not used.

74381 **INPUT FILES**

74382 None.

74383 **ENVIRONMENT VARIABLES**

74384 None.

74385 **ASYNCHRONOUS EVENTS**

74386 Default.

74387 **STDOUT**

74388 See the DESCRIPTION.

74389 **STDERR**

74390 The standard error shall be used only for diagnostic messages.

74391 **OUTPUT FILES**

74392 None.

74393 **EXTENDED DESCRIPTION**

74394 None.

74395 **EXIT STATUS**

74396 Zero.

74397 **CONSEQUENCES OF ERRORS**

74398 Default.

74399 **APPLICATION USAGE**

74400 None.

74401 **EXAMPLES**74402 `readonly HOME PWD`74403 **RATIONALE**74404 Some historical shells preserve the *readonly* attribute across separate invocations. This volume of
74405 POSIX.1-2008 allows this behavior, but does not require it.74406 The `-p` option allows portable access to the values that can be saved and then later restored
74407 using, for example, a *dot* script. Also see the RATIONALE for *export* (on page 2349) for a
74408 description of the no-argument and `-p` output cases and a related example.74409 Read-only functions were considered, but they were omitted as not being historical practice or
74410 particularly useful. Furthermore, functions must not be read-only across invocations to preclude
74411 “spoofing” (spoofing is the term for the practice of creating a program that acts like a well-
74412 known utility with the intent of subverting the real intent of the user) of administrative or
74413 security-relevant (or security-conscious) shell scripts.74414 **FUTURE DIRECTIONS**

74415 None.

74416 **SEE ALSO**74417 [Section 2.14](#) (on page 2334)74418 XBD [Section 12.2](#) (on page 215)74419 **CHANGE HISTORY**74420 **Issue 6**

74421 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

74422 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
74423 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
74424 behavior is intended.74425 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/7 is applied, adding the following text to
74426 the end of the first paragraph of the DESCRIPTION: “If the name of a variable is followed by
74427 *=word*, then the value of that variable shall be set to *word*.”. The reason for this change is that the
74428 SYNOPSIS for *readonly* includes:

readonly*Shell Command Language*

74429 `readonly name[=word] . . .`

74430 but the meaning of the optional “=word” is never explained in the text.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Shell Command Language

return74431 **NAME**

74432 return — return from a function

74433 **SYNOPSIS**74434 return [*n*]74435 **DESCRIPTION**74436 The *return* utility shall cause the shell to stop executing the current function or *dot* script. If the
74437 shell is not currently executing a function or *dot* script, the results are unspecified.74438 **OPTIONS**

74439 None.

74440 **OPERANDS**

74441 See the DESCRIPTION.

74442 **STDIN**

74443 Not used.

74444 **INPUT FILES**

74445 None.

74446 **ENVIRONMENT VARIABLES**

74447 None.

74448 **ASYNCHRONOUS EVENTS**

74449 Default.

74450 **STDOUT**

74451 Not used.

74452 **STDERR**

74453 The standard error shall be used only for diagnostic messages.

74454 **OUTPUT FILES**

74455 None.

74456 **EXTENDED DESCRIPTION**

74457 None.

74458 **EXIT STATUS**74459 The value of the special parameter '?' shall be set to *n*, an unsigned decimal integer, or to the
74460 exit status of the last command executed if *n* is not specified. If the value of *n* is greater than 255,
74461 the results are undefined. When *return* is executed in a *trap* action, the last command is
74462 considered to be the command that executed immediately preceding the *trap* action.74463 **CONSEQUENCES OF ERRORS**

74464 Default.

74465 **APPLICATION USAGE**

74466 None.

74467 **EXAMPLES**

74468 None.

74469 **RATIONALE**74470 The behavior of *return* when not in a function or *dot* script differs between the System V shell
74471 and the KornShell. In the System V shell this is an error, whereas in the KornShell, the effect is
74472 the same as *exit*.

74473 The results of returning a number greater than 255 are undefined because of differing practices

return*Shell Command Language*

74474 in the various historical implementations. Some shells AND out all but the low-order 8 bits;
74475 others allow larger values, but not of unlimited size.

74476 See the discussion of appropriate exit status values under *exit* (on page 2347).

74477 FUTURE DIRECTIONS

74478 None.

74479 SEE ALSO

74480 [Section 2.14](#) (on page 2334)

74481 CHANGE HISTORY**74482 Issue 6**

74483 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
74484 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
74485 behavior is intended.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

set — set or unset options and positional parameters

SYNOPSIS

set [-abCefhmnuvx] [-o option] [argument...]

set [+abCefhmnuvx] [+o option] [argument...]

set -- [argument...]

set -o

set +o

DESCRIPTION

If no *options* or *arguments* are specified, *set* shall write the names and values of all shell variables in the collation sequence of the current locale. Each *name* shall start on a separate line, using the format:

"%s=%s\n", <name>, <value>

The *value* string shall be written with appropriate quoting; see the description of shell quoting in [Section 2.2](#) (on page 2298). The output shall be suitable for reinput to the shell, setting or resetting, as far as possible, the variables that are currently set; read-only variables cannot be reset.

When options are specified, they shall set or unset attributes of the shell, as described below. When *arguments* are specified, they cause positional parameters to be set or unset, as described below. Setting or unsetting attributes and positional parameters are not necessarily related actions, but they can be combined in a single invocation of *set*.

The *set* special built-in shall support XBD [Section 12.2](#) (on page 215) except that options can be specified with either a leading <hyphen> (meaning enable the option) or <plus-sign> (meaning disable it) unless otherwise specified.

Implementations shall support the options in the following list in both their <hyphen> and <plus-sign> forms. These options can also be specified as options to *sh*.

-a When this option is on, the *export* attribute shall be set for each variable to which an assignment is performed; see XBD [Section 4.22](#) (on page 118). If the assignment precedes a utility name in a command, the *export* attribute shall not persist in the current execution environment after the utility completes, with the exception that preceding one of the special built-in utilities causes the *export* attribute to persist after the built-in has completed. If the assignment does not precede a utility name in the command, or if the assignment is a result of the operation of the *getopts* or *read* utilities, the *export* attribute shall persist until the variable is unset.

-b This option shall be supported if the implementation supports the User Portability Utilities option. It shall cause the shell to notify the user asynchronously of background job completions. The following message is written to standard error:

"[%d] %c %s\n", <job-number>, <current>, <status>, <job-name>

where the fields shall be as follows:

<current>	The character '+' identifies the job that would be used as a default for the <i>fg</i> or <i>bg</i> utilities; this job can also be specified using the <i>job_id</i> "%+" or "%%". The character '-' identifies the job that would become the default if the current default job were to exit; this job can also be specified using the <i>job_id</i> "%-". For other jobs, this field is a <space>. At most one job

set

Shell Command Language

- 74530 can be identified with '+' and at most one job can be identified with '-'.
 74531 If there is any suspended job, then the current job shall be a suspended
 74532 job. If there are at least two suspended jobs, then the previous job also
 74533 shall be a suspended job.
- 74534 <job-number> A number that can be used to identify the process group to the *wait*, *fg*, *bg*,
 74535 and *kill* utilities. Using these utilities, the job can be identified by prefixing
 74536 the job number with '%'.
 74537 <status> Unspecified.
 74538 <job-name> Unspecified.
- 74539 When the shell notifies the user a job has been completed, it may remove the job's process
 74540 ID from the list of those known in the current shell execution environment; see [Section](#)
 74541 [2.9.3.1](#) (on page 2319). Asynchronous notification shall not be enabled by default.
- 74542 -C (Uppercase C.) Prevent existing files from being overwritten by the shell's '>' redirection
 74543 operator (see [Section 2.7.2](#), on page 2313); the '>| ' redirection operator shall override this
 74544 *noclobber* option for an individual file.
- 74545 -e When this option is on, if a simple command fails for any of the reasons listed in [Section](#)
 74546 [2.8.1](#) (on page 2315) or returns an exit status value >0, and is not part of the compound list
 74547 following a **while**, **until**, or **if** keyword, and is not a part of an AND or OR list, and is not a
 74548 pipeline preceded by the ! reserved word, then the shell shall immediately exit.
- 74549 -f The shell shall disable pathname expansion.
- 74550 -h Locate and remember utilities invoked by functions as those functions are defined (the
 74551 utilities are normally located when the function is executed).
- 74552 -m This option shall be supported if the implementation supports the User Portability Utilities
 74553 option. All jobs shall be run in their own process groups. Immediately before the shell issues
 74554 a prompt after completion of the background job, a message reporting the exit status of the
 74555 background job shall be written to standard error. If a foreground job stops, the shell shall
 74556 write a message to standard error to that effect, formatted as described by the *jobs* utility. In
 74557 addition, if a job changes status other than exiting (for example, if it stops for input or
 74558 output or is stopped by a SIGSTOP signal), the shell shall write a similar message
 74559 immediately prior to writing the next prompt. This option is enabled by default for
 74560 interactive shells.
- 74561 -n The shell shall read commands but does not execute them; this can be used to check for
 74562 shell script syntax errors. An interactive shell may ignore this option.
- 74563 -o Write the current settings of the options to standard output in an unspecified format.
- 74564 +o Write the current option settings to standard output in a format that is suitable for reinput
 74565 to the shell as commands that achieve the same options settings.
- 74566 -o *option*
 74567 This option is supported if the system supports the User Portability Utilities option. It shall
 74568 set various options, many of which shall be equivalent to the single option letters. The
 74569 following values of *option* shall be supported:
- 74570 *allexport* Equivalent to -a.
 74571 *errexit* Equivalent to -e.

Shell Command Language

set

74572	<i>ignoreeof</i>	Prevent an interactive shell from exiting on end-of-file. This setting prevents accidental logouts when <control>-D is entered. A user shall explicitly <i>exit</i> to leave the interactive shell.
74573		
74574		
74575	<i>monitor</i>	Equivalent to -m . This option is supported if the system supports the User Portability Utilities option.
74576		
74577	<i>noclobber</i>	Equivalent to -C (uppercase C).
74578	<i>noglob</i>	Equivalent to -f .
74579	<i>noexec</i>	Equivalent to -n .
74580	<i>nolog</i>	Prevent the entry of function definitions into the command history; see Command History List (on page 3167).
74581		
74582	<i>notify</i>	Equivalent to -b .
74583	<i>nounset</i>	Equivalent to -u .
74584	<i>verbose</i>	Equivalent to -v .
74585	<i>vi</i>	Allow shell command line editing using the built-in <i>vi</i> editor. Enabling <i>vi</i> mode shall disable any other command line editing mode provided as an implementation extension.
74586		
74587		
74588		It need not be possible to set <i>vi</i> mode on for certain block-mode terminals.
74589	<i>xtrace</i>	Equivalent to -x .
74590	-u	The shell shall write a message to standard error when it tries to expand a variable that is not set and immediately exit. An interactive shell shall not exit.
74591		
74592	-v	The shell shall write its input to standard error as it is read.
74593	-x	The shell shall write to standard error a trace for each command after it expands the command and before it executes it. It is unspecified whether the command that turns tracing off is traced.
74594		
74595		
74596		The default for all these options shall be off (unset) unless stated otherwise in the description of the option or unless the shell was invoked with them on; see <i>sh</i> .
74597		
74598		The remaining arguments shall be assigned in order to the positional parameters. The special parameter ' # ' shall be set to reflect the number of positional parameters. All positional parameters shall be unset before any new values are assigned.
74599		
74600		
74601		If the first argument is ' - ', the results are unspecified.
74602		The special argument " -- " immediately following the <i>set</i> command name can be used to delimit the arguments if the first argument begins with ' + ' or ' - ', or to prevent inadvertent listing of all shell variables when there are no arguments. The command <i>set --</i> without <i>argument</i> shall unset all positional parameters and set the special parameter ' # ' to zero.
74603		
74604		
74605		

OPTIONS

See the DESCRIPTION.

OPERANDS

See the DESCRIPTION.

set*Shell Command Language*74610 **STDIN**

74611 Not used.

74612 **INPUT FILES**

74613 None.

74614 **ENVIRONMENT VARIABLES**

74615 None.

74616 **ASYNCHRONOUS EVENTS**

74617 Default.

74618 **STDOUT**

74619 See the DESCRIPTION.

74620 **STDERR**

74621 The standard error shall be used only for diagnostic messages.

74622 **OUTPUT FILES**

74623 None.

74624 **EXTENDED DESCRIPTION**

74625 None.

74626 **EXIT STATUS**

74627 Zero.

74628 **CONSEQUENCES OF ERRORS**

74629 Default.

74630 **APPLICATION USAGE**

74631 None.

74632 **EXAMPLES**

74633 Write out all variables and their values:

74634 `set`

74635 Set \$1, \$2, and \$3 and set "\$#" to 3:

74636 `set c a b`74637 Turn on the `-x` and `-v` options:74638 `set -xv`

74639 Unset all positional parameters:

74640 `set --`74641 Set \$1 to the value of `x`, even if it begins with `'-'` or `'+'`:74642 `set -- "$x"`74643 Set the positional parameters to the expansion of `x`, even if `x` expands with a leading `'-'` or `'+'`:74644 `set -- $x`74645 **RATIONALE**

74646 The `set --` form is listed specifically in the SYNOPSIS even though this usage is implied by the
 74647 Utility Syntax Guidelines. The explanation of this feature removes any ambiguity about whether
 74648 the `set --` form might be misinterpreted as being equivalent to `set` without any options or
 74649 arguments. The functionality of this form has been adopted from the KornShell. In System V, `set`

-- only unsets parameters if there is at least one argument; the only way to unset all parameters is to use *shift*. Using the KornShell version should not affect System V scripts because there should be no reason to issue it without arguments deliberately; if it were issued as, for example:

```
set -- "$@"
```

and there were in fact no arguments resulting from "\$@", unsetting the parameters would have no result.

The *set* + form in early proposals was omitted as being an unnecessary duplication of *set* alone and not widespread historical practice.

The *noclobber* option was changed to allow *set -C* as well as the *set -o noclobber* option. The single-letter version was added so that the historical "\$-" paradigm would not be broken; see [Section 2.5.2](#) (on page 2302).

The *-h* flag is related to command name hashing. See [hash](#) (on page 2788).

The following *set* flags were omitted intentionally with the following rationale:

-k The *-k* flag was originally added by the author of the Bourne shell to make it easier for users of pre-release versions of the shell. In early versions of the Bourne shell the construct *set name=value* had to be used to assign values to shell variables. The problem with *-k* is that the behavior affects parsing, virtually precluding writing any compilers. To explain the behavior of *-k*, it is necessary to describe the parsing algorithm, which is implementation-defined. For example:

```
set -k; echo name=value
```

and:

```
set -k
echo name=value
```

behave differently. The interaction with functions is even more complex. What is more, the *-k* flag is never needed, since the command line could have been reordered.

-t The *-t* flag is hard to specify and almost never used. The only known use could be done with here-documents. Moreover, the behavior with *ksh* and *sh* differs. The reference page says that it exits after reading and executing one command. What is one command? If the input is *date;date*, *sh* executes both *date* commands while *ksh* does only the first.

Consideration was given to rewriting *set* to simplify its confusing syntax. A specific suggestion was that the *unset* utility should be used to unset options instead of using the non-*getopt*()-able *+option* syntax. However, the conclusion was reached that the historical practice of using *+option* was satisfactory and that there was no compelling reason to modify such widespread historical practice.

The *-o* option was adopted from the KornShell to address user needs. In addition to its generally friendly interface, *-o* is needed to provide the *vi* command line editing mode, for which historical practice yields no single-letter option name. (Although it might have been possible to invent such a letter, it was recognized that other editing modes would be developed and *-o* provides ample name space for describing such extensions.)

Historical implementations are inconsistent in the format used for *-o* option status reporting. The *+o* format without an option-argument was added to allow portable access to the options that can be saved and then later restored using, for instance, a dot script.

Historically, *sh* did trace the command *set +x*, but *ksh* did not.

set*Shell Command Language*

The *ignoreeof* setting prevents accidental logouts when the end-of-file character (typically <control>-D) is entered. A user shall explicitly *exit* to leave the interactive shell.

The *set -m* option was added to apply only to the UPE because it applies primarily to interactive use, not shell script applications.

The ability to do asynchronous notification became available in the 1988 version of the KornShell. To have it occur, the user had to issue the command:

```
trap "jobs -n" CLD
```

The C shell provides two different levels of an asynchronous notification capability. The environment variable *notify* is analogous to what is done in *set -b* or *set -o notify*. When set, it notifies the user immediately of background job completions. When unset, this capability is turned off.

The other notification ability comes through the built-in utility *notify*. The syntax is:

```
notify [%job ... ]
```

By issuing *notify* with no operands, it causes the C shell to notify the user asynchronously when the state of the current job changes. If given operands, *notify* asynchronously informs the user of changes in the states of the specified jobs.

To add asynchronous notification to the POSIX shell, neither the KornShell extensions to *trap*, nor the C shell *notify* environment variable seemed appropriate (*notify* is not a proper POSIX environment variable name).

The *set -b* option was selected as a compromise.

The *notify* built-in was considered to have more functionality than was required for simple asynchronous notification.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.14](#) (on page 2334), *hash*

XBD [Section 4.22](#) (on page 118), [Section 12.2](#) (on page 215)

CHANGE HISTORY**Issue 6**

The obsolescent *set* command name followed by ' - ' has been removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *nolog* option is added to *set -o*.

IEEE PASC Interpretation 1003.2 #167 is applied, clarifying that the options default also takes into account the description of the option.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in behavior is intended.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/8 is applied, changing the square brackets in the example in RATIONALE to be in bold, which is the typeface used for optional items.

Issue 7

- 74734 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first
74735 argument is `'-'`.
74736
- 74737 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
74738 XSI shading is removed from the `-h` functionality.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

shift*Shell Command Language*74739 **NAME**

74740 shift — shift positional parameters

74741 **SYNOPSIS**74742 shift [*n*]74743 **DESCRIPTION**

74744 The positional parameters shall be shifted. Positional parameter 1 shall be assigned the value of
 74745 parameter (1+*n*), parameter 2 shall be assigned the value of parameter (2+*n*), and so on. The
 74746 parameters represented by the numbers "\$#" down to "\$#-*n*+1" shall be unset, and the
 74747 parameter '#' is updated to reflect the new number of positional parameters.

74748 The value *n* shall be an unsigned decimal integer less than or equal to the value of the special
 74749 parameter '#'. If *n* is not given, it shall be assumed to be 1. If *n* is 0, the positional and special
 74750 parameters are not changed.

74751 **OPTIONS**

74752 None.

74753 **OPERANDS**

74754 See the DESCRIPTION.

74755 **STDIN**

74756 Not used.

74757 **INPUT FILES**

74758 None.

74759 **ENVIRONMENT VARIABLES**

74760 None.

74761 **ASYNCHRONOUS EVENTS**

74762 Default.

74763 **STDOUT**

74764 Not used.

74765 **STDERR**

74766 The standard error shall be used only for diagnostic messages.

74767 **OUTPUT FILES**

74768 None.

74769 **EXTENDED DESCRIPTION**

74770 None.

74771 **EXIT STATUS**74772 The exit status is >0 if *n*>\$#; otherwise, it is zero.74773 **CONSEQUENCES OF ERRORS**

74774 Default.

74775 **APPLICATION USAGE**

74776 None.

74777 **EXAMPLES**

74778 \$ set a b c d e

74779 \$ shift 2

74780 \$ echo \$*

74781 c d e

74782 **RATIONALE**

74783 None.

74784 **FUTURE DIRECTIONS**

74785 None.

74786 **SEE ALSO**74787 [Section 2.14](#) (on page 2334)74788 **CHANGE HISTORY**74789 **Issue 6**

74790 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
74791 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
74792 behavior is intended.

times

Shell Command Language

74793 **NAME**74794 `times` — write process times74795 **SYNOPSIS**74796 `times`74797 **DESCRIPTION**

74798 The *times* utility shall write the accumulated user and system times for the shell and for all of its
 74799 child processes, in the following POSIX locale format:

74800 `"%dm%fs %dm%fs\n%dm%fs %dm%fs\n", <shell user minutes>,
 74801 <shell user seconds>, <shell system minutes>,
 74802 <shell system seconds>, <children user minutes>,
 74803 <children user seconds>, <children system minutes>,
 74804 <children system seconds>`

74805 The four pairs of times shall correspond to the members of the `<sys/times.h>` `tms` structure
 74806 (defined in XBD Chapter 13, on page 219) as returned by `times()`: `tms_utime`, `tms_stime`,
 74807 `tms_cutime`, and `tms_cstime`, respectively.

74808 **OPTIONS**

74809 None.

74810 **OPERANDS**

74811 None.

74812 **STDIN**

74813 Not used.

74814 **INPUT FILES**

74815 None.

74816 **ENVIRONMENT VARIABLES**

74817 None.

74818 **ASYNCHRONOUS EVENTS**

74819 Default.

74820 **STDOUT**

74821 See the DESCRIPTION.

74822 **STDERR**

74823 The standard error shall be used only for diagnostic messages.

74824 **OUTPUT FILES**

74825 None.

74826 **EXTENDED DESCRIPTION**

74827 None.

74828 **EXIT STATUS**

74829 Zero.

74830 **CONSEQUENCES OF ERRORS**

74831 Default.

74832 **APPLICATION USAGE**

74833 None.

74834 **EXAMPLES**74835 `$ times`74836 `0m0.43s 0m1.11s`74837 `8m44.18s 1m43.23s`74838 **RATIONALE**74839 The *times* special built-in from the Single UNIX Specification is now required for all conforming
74840 shells.74841 **FUTURE DIRECTIONS**

74842 None.

74843 **SEE ALSO**74844 [Section 2.14](#) (on page 2334)74845 XBD [<sys/times.h>](#)74846 **CHANGE HISTORY**74847 **Issue 6**74848 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/9 is applied, changing text in the
74849 DESCRIPTION from: "Write the accumulated user and system times for the shell and for all of
74850 its child processes ..." to: "The *times* utility shall write the accumulated user and system times for
74851 the shell and for all of its child processes ...".

trap*Shell Command Language*74852 **NAME**74853 `trap` — trap signals74854 **SYNOPSIS**74855 `trap n [condition...]`74856 `trap [action condition...]`74857 **DESCRIPTION**

74858 If the first operand is an unsigned decimal integer, the shell shall treat all operands as
 74859 conditions, and shall reset each condition to the default value. Otherwise, if there are operands,
 74860 the first is treated as an action and the remaining as conditions.

74861 If *action* is `'-'`, the shell shall reset each *condition* to the default value. If *action* is null (`" "`), the
 74862 shell shall ignore each specified *condition* if it arises. Otherwise, the argument *action* shall be read
 74863 and executed by the shell when one of the corresponding conditions arises. The action of *trap*
 74864 shall override a previous action (either default action or one explicitly set). The value of `"$?"`
 74865 after the *trap* action completes shall be the value it had before *trap* was invoked.

74866 The condition can be EXIT, 0 (equivalent to EXIT), or a signal specified using a symbolic name,
 74867 without the SIG prefix, as listed in the tables of signal names in the `<signal.h>` header defined in
 74868 XBD Chapter 13 (on page 219); for example, HUP, INT, QUIT, TERM. Implementations may
 74869 permit names with the SIG prefix or ignore case in signal names as an extension. Setting a trap
 74870 for SIGKILL or SIGSTOP produces undefined results.

74871 The environment in which the shell executes a *trap* on EXIT shall be identical to the environment
 74872 immediately after the last command executed before the *trap* on EXIT was taken.

74873 Each time *trap* is invoked, the *action* argument shall be processed in a manner equivalent to:

74874 `eval action`

74875 Signals that were ignored on entry to a non-interactive shell cannot be trapped or reset, although
 74876 no error need be reported when attempting to do so. An interactive shell may reset or catch
 74877 signals ignored on entry. Traps shall remain in place for a given shell until explicitly changed
 74878 with another *trap* command.

74879 When a subshell is entered, traps that are not being ignored are set to the default actions. This
 74880 does not imply that the *trap* command cannot be used within the subshell to set new traps.

74881 The *trap* command with no arguments shall write to standard output a list of commands
 74882 associated with each condition. The format shall be:

74883 `"trap -- %s %s ...\n", <action>, <condition> ...`

74884 The shell shall format the output, including the proper use of quoting, so that it is suitable for
 74885 reinput to the shell as commands that achieve the same trapping results. For example:

74886 `save_traps=$(trap)`74887 `...`74888 `eval "$save_traps"`

74889 XSI the *trap* command with no arguments shall write to standard output a list of commands
 74890 associated with each condition. The format shall be:

74891 `1 SIGHUP`74892 `2 SIGINT`74893 `3 SIGQUIT`

74894	6	SIGABRT
74895	9	SIGKILL
74896	14	SIGALRM
74897	15	SIGTERM

74898 The *trap* special built-in shall conform to XBD [Section 12.2](#) (on page 215).

74899 OPTIONS

74900 None.

74901 OPERANDS

74902 See the DESCRIPTION.

74903 STDIN

74904 Not used.

74905 INPUT FILES

74906 None.

74907 ENVIRONMENT VARIABLES

74908 None.

74909 ASYNCHRONOUS EVENTS

74910 Default.

74911 STDOUT

74912 See the DESCRIPTION.

74913 STDERR

74914 The standard error shall be used only for diagnostic messages.

74915 OUTPUT FILES

74916 None.

74917 EXTENDED DESCRIPTION

74918 None.

74919 EXIT STATUS

74920 XSI If the trap name or number is invalid, a non-zero exit status shall be returned; otherwise, zero
 74921 XSI shall be returned. For both interactive and non-interactive shells, invalid signal names or
 74922 numbers shall not be considered a syntax error and do not cause the shell to abort.

74923 CONSEQUENCES OF ERRORS

74924 Default.

74925 APPLICATION USAGE

74926 None.

74927 EXAMPLES

74928 Write out a list of all traps and actions:

74929 trap

74930 Set a trap so the *logout* utility in the directory referred to by the *HOME* environment variable
 74931 executes when the shell terminates:

74932 trap '\$HOME/logout' EXIT

74933 or:

trap*Shell Command Language*

74934 `trap '$HOME/logout' 0`

74935 Unset traps on INT, QUIT, TERM, and EXIT:

74936 `trap - INT QUIT TERM EXIT`

74937 **RATIONALE**

74938 Implementations may permit lowercase signal names as an extension. Implementations may
 74939 also accept the names with the SIG prefix; no known historical shell does so. The *trap* and *kill*
 74940 utilities in this volume of POSIX.1-2008 are now consistent in their omission of the SIG prefix for
 74941 signal names. Some *kill* implementations do not allow the prefix, and *kill -l* lists the signals
 74942 without prefixes.

74943 Trapping SIGKILL or SIGSTOP is syntactically accepted by some historical implementations, but
 74944 it has no effect. Portable POSIX applications cannot attempt to trap these signals.

74945 The output format is not historical practice. Since the output of historical *trap* commands is not
 74946 portable (because numeric signal values are not portable) and had to change to become so, an
 74947 opportunity was taken to format the output in a way that a shell script could use to save and
 74948 then later reuse a trap if it wanted.

74949 The KornShell uses an **ERR** trap that is triggered whenever *set -e* would cause an exit. This is
 74950 allowable as an extension, but was not mandated, as other shells have not used it.

74951 The text about the environment for the EXIT trap invalidates the behavior of some historical
 74952 versions of interactive shells which, for example, close the standard input before executing a
 74953 trap on 0. For example, in some historical interactive shell sessions the following trap on 0
 74954 would always print "--":

74955 `trap 'read foo; echo "--$foo--"' 0`

74956 The command:

74957 `trap '$cmd' 0`

74958 causes the contents of the shell variable *cmd* to be executed as a command when the shell exits.
 74959 Using double-quotes instead of single-quotes might have unexpected behavior, since in theory
 74960 the value of *cmd* might be a decimal integer which would be treated as a condition, not an
 74961 action; or *cmd* might begin with '-'. Also, using double-quotes will cause the value of *cmd* to be
 74962 expanded twice, once when *trap* is executed, and once when the condition arises.

74963 **FUTURE DIRECTIONS**

74964 None.

74965 **SEE ALSO**

74966 [Section 2.14](#) (on page 2334)

74967 [XBD Section 12.2](#) (on page 215), [<signal.h>](#)

74968 **CHANGE HISTORY**

74969 **Issue 6**

74970 XSI-conforming implementations provide the mapping of signal names to numbers given above
 74971 (previously this had been marked obsolescent). Other implementations need not provide this
 74972 optional mapping.

74973 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
 74974 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
 74975 behavior is intended.

Issue 7

74976 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

74977 Austin Group Interpretation 1003.1-2001 #116 is applied.

74978

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

unset*Shell Command Language*74979 **NAME**

74980 unset — unset values and attributes of variables and functions

74981 **SYNOPSIS**74982 unset [-fv] *name*...74983 **DESCRIPTION**74984 Each variable or function specified by *name* shall be unset.74985 If **-v** is specified, *name* refers to a variable name and the shell shall unset it and remove it from
74986 the environment. Read-only variables cannot be unset.74987 If **-f** is specified, *name* refers to a function and the shell shall unset the function definition.74988 If neither **-f** nor **-v** is specified, *name* refers to a variable; if a variable by that name does not
74989 exist, it is unspecified whether a function by that name, if any, shall be unset.74990 Unsetting a variable or function that was not previously set shall not be considered an error and
74991 does not cause the shell to abort.74992 The *unset* special built-in shall support XBD [Section 12.2](#) (on page 215).

74993 Note that:

74994 VARIABLE=

74995 is not equivalent to an *unset* of **VARIABLE**; in the example, **VARIABLE** is set to ". Also, the
74996 variables that can be *unset* should not be misinterpreted to include the special parameters (see
74997 [Section 2.5.2](#), on page 2302).74998 **OPTIONS**

74999 See the DESCRIPTION.

75000 **OPERANDS**

75001 See the DESCRIPTION.

75002 **STDIN**

75003 Not used.

75004 **INPUT FILES**

75005 None.

75006 **ENVIRONMENT VARIABLES**

75007 None.

75008 **ASYNCHRONOUS EVENTS**

75009 Default.

75010 **STDOUT**

75011 Not used.

75012 **STDERR**

75013 The standard error shall be used only for diagnostic messages.

75014 **OUTPUT FILES**

75015 None.

75016 **EXTENDED DESCRIPTION**

75017 None.

75018 **EXIT STATUS**75019 0 All *name* operands were successfully unset.75020 >0 At least one *name* could not be unset.75021 **CONSEQUENCES OF ERRORS**

75022 Default.

75023 **APPLICATION USAGE**

75024 None.

75025 **EXAMPLES**75026 Unset *VISUAL* variable:75027 unset -v *VISUAL*75028 Unset the functions **foo** and **bar**:

75029 unset -f foo bar

75030 **RATIONALE**75031 Consideration was given to omitting the **-f** option in favor of an *unfunction* utility, but the
75032 standard developers decided to retain historical practice.75033 The **-v** option was introduced because System V historically used one name space for both
75034 variables and functions. When *unset* is used without options, System V historically unset either a
75035 function or a variable, and there was no confusion about which one was intended. A portable
75036 POSIX application can use *unset* without an option to unset a variable, but not a function; the **-f**
75037 option must be used.75038 **FUTURE DIRECTIONS**

75039 None.

75040 **SEE ALSO**75041 [Section 2.14](#) (on page 2334)75042 XBD [Section 12.2](#) (on page 215)75043 **CHANGE HISTORY**75044 **Issue 6**75045 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
75046 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
75047 behavior is intended.75048 **Issue 7**

75049 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

(blank page)

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Batch Environment Services

OB BE

This chapter describes the services and utilities that shall be implemented on all systems that claim conformance to the Batch Environment Services and Utilities option. The functionality described in this section shall be provided on implementations that support the Batch Environment Services and Utilities option (and the rest of this section is not further shaded for this option).

Note that the Batch Environment Services and Utilities option is marked obsolescent in Issue 7.

3.1 General Concepts**3.1.1 Batch Client-Server Interaction**

Batch jobs are created and managed by batch servers. A batch client interacts with a batch server to access batch services on behalf of the user. In order to use batch services, a user must have access to a batch client.

A batch server is a computational entity, such as a daemon process, that provides batch services. Batch servers route, queue, modify, and execute batch jobs on behalf of batch clients.

The batch utilities described in this volume of POSIX.1-2008 (and listed in [Table 3-1](#)) are clients of batch services; they allow users to perform actions on the job such as creating, modifying, and deleting batch jobs from a shell command line. Although these batch utilities may be said to accomplish certain services, they actually obtain services on behalf of a user by means of requests to batch servers.

Table 3-1 Batch Utilities

<i>qalter</i>	<i>qmove</i>	<i>qrls</i>	<i>qstat</i>
<i>qdel</i>	<i>qmsg</i>	<i>qselect</i>	<i>qsub</i>
<i>qhold</i>	<i>qrerun</i>	<i>qsig</i>	

Client-server interaction takes place by means of the batch requests defined in this chapter. Because direct access to batch jobs and queues is limited to batch servers, clients and servers of different implementations can interoperate, since dependencies on private structures for batch jobs and queues are limited to batch servers. Also, batch servers may be clients of other batch servers.

3.1.2 Batch Queues

Two types of batch queue are described: routing queues and execution queues. When a batch job is placed in a routing queue, it is a candidate for routing. A batch job is removed from routing queues under the following conditions:

- The batch job has been routed to another queue.
- The batch job has been deleted from the batch queue.
- The batch job has been aborted.

When a batch job is placed in an execution queue, it is a candidate for execution.

A batch job is removed from an execution queue under the following conditions:

- The batch job has been executed and exited.
- The batch job has been aborted.
- The batch job has been deleted from the batch queue.
- The batch job has been moved to another queue.

Access to a batch queue is limited to the batch server that manages the batch queue. Clients never access a batch queue or a batch job directly, either to read or write information; all client access to batch queues or jobs takes place through batch servers.

3.1.3 Batch Job Creation

When a batch server creates a batch job on behalf of a client, it shall assign a batch job identifier to the job. A batch job identifier consists of both a sequence number that is unique among the sequence numbers issued by that server and the name of the server. Since the batch server name is unique within a name space, the job identifier is likewise unique within the name space.

The batch server that creates a batch job shall return the batch server-assigned job identifier to the client that requested the job creation. If the batch server routes or moves the job to another server, it sends the job identifier with the job. Once assigned, the job identifier of a batch job shall never change.

3.1.4 Batch Job Tracking

Since a batch job may be moved after creation, the batch server name component of the job identifier need not indicate the location of the job. An implementation may provide a batch job tracking mechanism, in which case the user generally does not need to know the location of the job. However, an implementation need not provide a batch job tracking mechanism, in which case the user must find routed jobs by probing the possible destinations.

3.1.5 Batch Job Routing

To route a batch job, a batch server either moves the job to some other queue that is managed by the batch server, or requests that some other batch server accept the job.

Each routing queue has one or more queues to which it can route batch jobs. The batch server administrator creates routing queues.

A batch server may route a batch job from a routing queue to another routing queue. Batch servers shall prevent or otherwise handle cases of circular routing paths. As a deferred service, a batch server routes jobs from the routing queues that it manages. The algorithm by which a batch server selects a batch queue to which to route a batch job is implementation-defined.

A batch job need not be eligible for routing to all the batch queues fed by the routing queue from which it is routed. A batch server that has been asked to accept the job may reject the request if the job requires resources that are unavailable to that batch server, or if the client is not authorized to access the batch server.

Batch servers may route high-priority jobs before low-priority jobs, but, on other than overloaded systems, the effect may be imperceptible to the user. If all the batch servers fed by a routing queue reject requests to accept the job for reasons that are permanent, the batch server that manages the job shall abort the job. If all or some rejections are temporary, the batch server should try to route the job again at some later point.

The reasons for rejecting a batch job are implementation-defined.

The reasons for which the routing should be retried later and the reasons for which the job should be aborted are also implementation-defined.

3.1.6 Batch Job Execution

To execute a batch job is to create a session leader (a process) that runs the shell program indicated by the *Shell_Path* attribute of the job. The script shall be passed to the program as its standard input. An implementation may pass the script to the program by other implementation-defined means. At the time a batch job begins execution, it is defined to enter the RUNNING state. The primary program that is executed by a batch job is typically, though not necessarily, a shell program.

A batch server shall execute eligible jobs as a deferred service—no client request is necessary once the batch job is created and eligible. However, the attributes of a batch job, such as the job hold type, may render the job ineligible. A batch server shall scan the execution queues that it manages for jobs that are eligible for execution. The algorithm by which the batch server selects eligible jobs for execution is implementation-defined.

As part of creating the process for the batch job, the batch server shall open the standard output and standard error streams of the session.

The attributes of a batch job may indicate that the batch server executing the job shall send mail to a list of users at the time it begins execution of the job.

75147 **3.1.7 Batch Job Exit**

75148 When the session leader of an executing job terminates, the job exits. As part of exiting a batch
 75149 job, the batch server that manages the job shall remove the job from the batch queue in which it
 75150 resides. The server shall transfer output files of the job to a location described by the attributes of
 75151 the job.

75152 The attributes of a batch job may indicate that the batch server managing the job shall send mail
 75153 to a list of users at the time the job exits.

75154 **3.1.8 Batch Job Abort**

75155 A batch server shall abort jobs for which a required deferred service cannot be performed. The
 75156 attributes of a batch job may indicate that the batch server that aborts the job shall send mail to a
 75157 list of users at the time it aborts the job.

75158 **3.1.9 Batch Authorization**

75159 Clients, such as the batch environment utilities (marked BE), access batch services by means of
 75160 requests to one or more batch servers. To acquire the services of any given batch server, the user
 75161 identifier under which the client runs must be authorized to use that batch server.

75162 The user with an associated user name that creates a batch job shall own the job and can perform
 75163 actions such as read, modify, delete, and move.

75164 A user identifier of the same value at a different host need not be the same user. For example,
 75165 user name *smith* at host **alpha** may or may not represent the same person as user name *smith* at
 75166 host **beta**. Likewise, the same person may have access to different user names on different hosts.

75167 An implementation may optionally provide an authorization mechanism that permits one user
 75168 name to access jobs under another user name.

75169 A process on a client host may be authorized to run processes under multiple user names at a
 75170 batch server host. Where appropriate, the utilities defined in this volume of POSIX.1-2008
 75171 provide a means for a user to choose from among such user names when creating or modifying
 75172 a batch job.

75173 **3.1.10 Batch Administration**

75174 The processing of a batch job by a batch server is affected by the attributes of the job. The
 75175 processing of a batch job may also be affected by the attributes of the batch queue in which the
 75176 job resides and by the status of the batch server that manages the job. See also XBD [Chapter 3](#)
 75177 (on page 33) for batch definitions.

3.1.11 Batch Notification

Whereas batch servers are persistent entities, clients are often transient. For example, the *qsub* utility creates a batch job and exits. For this reason, batch servers notify users of batch job events by sending mail to the user that owns the job, or to other designated users.

3.2 Batch Services

The presence of Batch Environment Services and Utilities option services is indicated by the configuration variable `POSIX2_PBS`. A conforming batch server provides services as defined in this section.

A batch server shall provide batch services in two ways:

1. The batch server provides a service at the request of a client.
2. The batch server provides a deferred service as a result of a change in conditions monitored by the batch server.

If a batch server cannot complete a request, it shall reject the request. If a batch server cannot complete a deferred service for a batch job, the batch server shall abort the batch job. Table 3-2 is a summary of environment variables that shall be supported by an implementation of the batch server and utilities.

Table 3-2 Environment Variable Summary

Variable	Description
<code>PBS_DPREFIX</code>	Defines the directive prefix (see <i>qsub</i>)
<code>PBS_ENVIRONMENT</code>	Batch job is batch or interactive (see Section 3.2.2.1)
<code>PBS_JOBID</code>	The <i>job_identifier</i> attribute of job (see Section 3.2.3.8)
<code>PBS_JOBNAME</code>	The <i>job_name</i> attribute of job (see Section 3.2.3.8)
<code>PBS_O_HOME</code>	Defines the <i>HOME</i> of the batch client (see <i>qsub</i>)
<code>PBS_O_HOST</code>	Defines the host name of the batch client (see <i>qsub</i>)
<code>PBS_O_LANG</code>	Defines the <i>LANG</i> of the batch client (see <i>qsub</i>)
<code>PBS_O_LOGNAME</code>	Defines the <i>LOGNAME</i> of the batch client (see <i>qsub</i>)
<code>PBS_O_MAIL</code>	Defines the <i>MAIL</i> of the batch client (see <i>qsub</i>)
<code>PBS_O_PATH</code>	Defines the <i>PATH</i> of the batch client (see <i>qsub</i>)
<code>PBS_O_QUEUE</code>	Defines the submit queue of the batch client (see <i>qsub</i>)
<code>PBS_O_SHELL</code>	Defines the <i>SHELL</i> of the batch client (see <i>qsub</i>)
<code>PBS_O_TZ</code>	Defines the <i>TZ</i> of the batch client (see <i>qsub</i>)
<code>PBS_O_WORKDIR</code>	Defines the working directory of the batch client (see <i>qsub</i>)
<code>PBS_QUEUE</code>	Defines the initial execution queue (see Section 3.2.2.1)

3.2.1 Batch Job States

A batch job shall always be in one of the following states: QUEUED, RUNNING, HELD, WAITING, EXITING, or TRANSITING. The state of a batch job determines the types of requests that the batch server that manages the batch job can accept for the batch job. A batch server shall change the state of a batch job either in response to service requests from clients or as a result of deferred services, such as job execution or job routing.

A batch job that is in the QUEUED state resides in a queue but is still pending either execution or routing, depending on the queue type.

A batch server that queues a batch job in a routing queue shall put the batch job in the QUEUED state. A batch server that puts a batch job in an execution queue, but has not yet executed the batch job, shall put the batch job in the QUEUED state. A batch job that resides in an execution queue and is executing is defined to be in the RUNNING state. While a batch job is in the RUNNING state, a session leader is associated with the batch job.

A batch job that resides in an execution queue, but is ineligible to run because of a hold attribute, is defined to be in the HELD state.

A batch job that is not held, but must wait until a future date and time before executing, is defined to be in the WAITING state.

When the session leader associated with a running job exits, the batch job shall be placed in the EXITING state.

A batch job for which the session leader has terminated is defined to be in the EXITING state, and the batch server that manages such a batch job cannot accept job modification requests that affect the batch job. While a batch job is in the EXITING state, the batch server that manages the batch job is staging output files and notifying clients of job completion. Once a batch job has exited, it no longer exists as an object managed by a batch server.

A batch job that is being moved from a routing queue to another queue is defined to be in the TRANSITING state.

When a batch job in a routing queue has been selected to be moved to a new destination, then the batch job shall be in either the QUEUED state or the TRANSITING state, depending on the batch server implementation.

Batch jobs with either an *Execution_Time* attribute value set in the future or a *Hold_Types* attribute of value not equal to NO_HOLD, or both, may be routed or held in the routing queue. The treatment of jobs with the *Execution_Time* or *Hold_Types* attributes in a routing queue is implementation-defined.

When a batch job in a routing queue has not been selected to be moved to a new destination and the batch job has a *Hold_Types* attribute value of other than NO_HOLD, then the job should be in the HELD state.

Note: The effect of a hold upon a batch job in a routing queue is implementation-defined. The implementation should use the state that matches whether the batch job can route with a hold or not.

When a batch job in a routing queue has not been selected to be moved to a new destination and the batch job has:

- A *Hold_Types* attribute value of NO_HOLD
- An *Execution_Time* attribute in the past

then the batch job shall be in the QUEUED state.

When a batch job in a routing queue has not been selected to be moved to a new destination and the batch job has:

- A *Hold_Types* attribute value of NO_HOLD
- An *Execution_Time* attribute in the future

then the batch job may be in the WAITING state.

Note: The effect of a future execution time upon a batch job in a routing queue is implementation-defined. The implementation should use the state that matches whether the batch job can route with a hold or not.

Table 3-3 describes the next state of a batch job, given the current state of the batch job and the type of request. Table 3-4 (on page 2383) describes the response of a batch server to a request, given the current state of the batch job and the type of request.

3.2.2 Deferred Batch Services

This section describes the deferred services performed by batch servers: job execution, job routing, job exit, job abort, and the rerunning of jobs after a restart.

3.2.2.1 Batch Job Execution

To execute a batch job is to create a session leader (a process) that runs the shell program indicated by the *Shell_Path_List* attribute of the batch job. The script is passed to the program as its standard input. An implementation may pass the script to the program by other implementation-defined means. At the time a batch job begins execution, it is defined to enter the RUNNING state.

Table 3-3 Next State Table

Request Type	Current State						
	X	Q	R	H	W	E	T
Queue Batch Job Request	Q	e	e	e	e	e	e
Modify Batch Job Request	e	Q	R	H	W	e	T
Delete Batch Job Request	e	X	E	X	X	E	X
Batch Job Message Request	e	Q	R	H	W	E	T
Rerun Batch Job Request	e	e	Q	e	e	e	e
Signal Batch Job Request	e	e	R	H	W	e	e
Batch Job Status Request	e	Q	R	H	W	E	T
Batch Queue Status Request	X	Q	R	H	W	E	T
Server Status Request	X	Q	R	H	W	E	T
Select Batch Jobs Request	X	Q	R	H	W	E	T
Move Batch Job Request	e	Q	R	H	W	e	T
Hold Batch Job Request	e	H	R/H	H	H	e	T
Release Batch Job Request	e	Q	R	Q/W/H	W	e	T
Server Shutdown Request	X	Q	Q	H	W	E	T
Locate Batch Job Request	e	Q	R	H	W	E	T

75293 **Legend**

75294 X Nonexistent

75295 Q QUEUED

75296 R RUNNING

75297 H HELD

75298 W WAITING

75299 E EXITING

75300 T TRANSITING

75301 e Error

75302 A batch server that has an execution queue containing jobs is said to own the queue and manage
 75303 the batch jobs in that queue. A batch server that has been started shall execute the batch jobs in
 75304 the execution queues owned by the batch server. The batch server shall schedule for execution
 75305 those jobs in the execution queues that are in the QUEUED state. The algorithm for scheduling
 75306 jobs is implementation-defined.

75307 A batch server that executes a batch job shall create, in the environment of the session leader of
 75308 the batch job, an environment variable named *PBS_ENVIRONMENT*, the value of which is the
 75309 string *PBS_BATCH* encoded in the portable character set.

75310 A batch server that executes a batch job shall create, in the environment of the session leader of
 75311 the batch job, an environment variable named *PBS_QUEUE*, the value of which is the name of
 75312 the execution queue of the batch job encoded in the portable character set.

75313 To rerun a batch job is to requeue a batch job that is currently executing and then kill the session
 75314 leader of the executing job by sending a SIGKILL prior to completion; see [Section 3.2.3.11](#) (on
 75315 page 2395). A batch server that reruns a batch job shall append the standard output and
 75316 standard error files of the batch job to the corresponding files of the previous execution, if they
 75317 exist, with appropriate annotation. If either file does not exist, that file shall be created as in
 75318 normal execution.

Table 3-4 Results/Output Table

Request Type	Current State						
	X	Q	R	H	W	E	T
Queue Batch Job Request	O	e	e	e	e	e	e
Modify Batch Job Request	e	O	e	O	O	e	e
Delete Batch Job Request	e	O	O	O	O	e	O
Batch Job Message Request	e	e	O	e	e	e	e
Rerun Batch Job Request	e	e	O	e	e	e	e
Signal Batch Job Request	e	e	O	e	e	e	e
Batch Job Status Request	e	O	O	O	O	O	O
Batch Queue Status Request	O	O	O	O	O	O	O
Server Status Request	O	O	O	O	O	O	O
Select Batch Job Request	e	O	O	O	O	O	O
Move Batch Job Request	e	O	O	O	O	e	e
Hold Batch Job Request	e	O	O	O	O	e	e
Release Batch Job Request	e	O	e	O	O	e	e
Server Shutdown Request	O	O	e	O	O	e	e
Locate Batch Job Request	e	O	O	O	O	O	O

Legend

O OK

e Error message

The execution of a batch job by a batch server shall be controlled by job, queue, and server attributes, as defined in this section.

Account_Name Attribute

Batch accounting is an optional feature of batch servers. If a batch server implements accounting, the statements in this section apply and the configuration variable POSIX2_PBS_ACCOUNTING shall be set to 1.

A batch server that executes a batch job shall charge the account named in the *Account_Name* attribute of the batch job for resources consumed by the batch job.

If the *Account_Name* attribute of the batch job is absent from the batch job attribute list or is altered while the batch job is in execution, the batch server action is implementation-defined.

Checkpoint Attribute

Batch checkpointing is an optional feature of batch servers. If a batch server implements checkpointing, the statements in this section apply and the configuration variable POSIX2_PBS_CHECKPOINT shall be set to 1.

There are two attributes associated with the checkpointing feature: *Checkpoint* and *Minimum_Cpu_Interval*. *Checkpoint* is a batch job attribute, while *Minimum_Cpu_Interval* is a queue attribute. An implementation that does not support checkpointing shall support the *Checkpoint* job attribute to the extent that the batch server shall maintain and pass this attribute to other servers.

The behavior of a batch server that executes a batch job for which the value of the *Checkpoint* attribute is CHECKPOINT_UNSPECIFIED is implementation-defined. A batch server that executes a batch job for which the value of the *Checkpoint* attribute is NO_CHECKPOINT shall

75362 not checkpoint the batch job.

75363 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is
75364 CHECKPOINT_AT_SHUTDOWN shall checkpoint the batch job only when the batch server
75365 accepts a request to shut down during the time when the batch job is in the RUNNING state.

75366 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is
75367 CHECKPOINT_AT_MIN_CPU_INTERVAL shall checkpoint the batch job at the interval
75368 specified by the *Minimum_Cpu_Interval* attribute of the queue for which the batch job has been
75369 selected. The *Minimum_Cpu_Interval* attribute shall be specified in units of CPU minutes.

75370 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is an
75371 unsigned integer shall checkpoint the batch job at an interval that is the value of either the
75372 *Checkpoint* attribute, or the *Minimum_Cpu_Interval* attribute of the queue for which the batch job
75373 has been selected, whichever is greater. Both intervals shall be in units of CPU minutes. When
75374 the *Minimum_Cpu_Interval* attribute is greater than the *Checkpoint* attribute, the batch job shall
75375 write a warning message to the standard error stream of the batch job.

75376 **Error_Path Attribute**

75377 The *Error_Path* attribute of a running job cannot be changed by a *Modify Batch Job Request*. When
75378 the *Join_Path* attribute of the batch job is set to the value FALSE and the *Keep_Files* attribute of
75379 the batch job does not contain the value KEEP_STD_ERROR, a batch server that executes a batch
75380 job shall perform one of the following actions:

- 75381 • Set the standard error stream of the session leader of the batch job to the path described by
75382 the value of the *Error_Path* attribute of the batch job.
- 75383 • Buffer the standard error of the session leader of the batch job until completion of the batch
75384 job, and when the batch job exits return the contents to the destination described by the
75385 value of the *Error_Path* attribute of the batch job.

75386 Applications shall not rely on having access to the standard error of a batch job prior to the
75387 completion of the batch job.

75388 When the *Error_Path* attribute does not specify a host name, then the batch server shall retain the
75389 standard error of the batch job on the host of execution.

75390 When the *Error_Path* attribute does specify a host name and the *Keep_Files* attribute does not
75391 contain the value KEEP_STD_ERROR, then the final destination of the standard error of the
75392 batch job shall be on the host whose host name is specified.

75393 If the path indicated by the value of the *Error_Path* attribute of the batch job is a relative path, the
75394 batch server shall expand the path relative to the home directory of the user on the host to which
75395 the file is being returned.

75396 When the batch server buffers the standard error of the batch job and the file cannot be opened
75397 for write upon completion of the batch job, then the server shall place the standard error in an
75398 implementation-defined location and notify the user of the location via mail. It shall be possible
75399 for the user to process this mail using the *mailx* utility.

75400 If a batch server that does not buffer the standard error cannot open the standard error path of
75401 the batch job for write access, then the batch server shall abort the batch job.

Execution_Time Attribute

A batch server shall not execute a batch job before the time represented by the value of the *Execution_Time* attribute of the batch job. The *Execution_Time* attribute is defined in seconds since the Epoch.

Hold_Types Attribute

A batch server shall support the following hold types:

- s Can be set or released by a user with at least a privilege level of batch administrator (SYSTEM).
- o Can be set or released by a user with at least a privilege level of batch operator (OPERATOR).
- u Can be set or released by the user with at least a privilege level of user, where the user is defined in the *Job_Owner* attribute (USER).
- n Indicates that none of the *Hold_Types* attributes are set (NO_HOLD).

An implementation may define other hold types. Any additional hold types, how they are specified, their internal representation, their behavior, and how they affect the behavior of other utilities are implementation-defined.

The value of the *Hold_Types* attribute shall be the union of the valid hold types ('s', 'o', 'u', and any implementation-defined hold types), or 'n'.

A batch server shall not execute a batch job if the *Hold_Types* attribute of the batch job has a value other than NO_HOLD. If the *Hold_Types* attribute of the batch job has a value other than NO_HOLD, the batch job shall be in the HELD state.

Job_Owner Attribute

The *Job_Owner* attribute consists of a pair of user name and host name values of the form:

username@hostname

A batch server that accepts a *Queue Batch Job Request* shall set the *Job_Owner* attribute to a string that is the *username@hostname* of the user who submitted the job.

Join_Path Attribute

A batch server that executes a batch job for which the value of the *Join_Path* attribute is TRUE shall ignore the value of the *Error_Path* attribute and merge the standard error of the batch job with the standard output of the batch job.

Keep_Files Attribute

A batch server that executes a batch job for which the value of the *Keep_Files* attribute includes the value KEEP_STD_OUTPUT shall retain the standard output of the batch job on the host where execution occurs. The standard output shall be retained in the home directory of the user under whose user ID the batch job is executed and the filename shall be the default filename for the standard output as defined under the *-o* option of the *qsub* utility. The *Output_Path* attribute is not modified.

A batch server that executes a batch job for which the value of the *Keep_Files* attribute includes the value KEEP_STD_ERROR shall retain the standard error of the batch job on the host where execution occurs. The standard error shall be retained in the home directory of the user under whose user ID the batch job is executed and the filename shall be the default filename for

75443 standard error as defined under the *-e* option of the *qsub* utility. The *Error_Path* attribute is not
75444 modified.

75445 A batch server that executes a batch job for which the value of the *Keep_Files* attribute includes
75446 values other than *KEEP_STD_OUTPUT* and *KEEP_STD_ERROR* shall retain these other files on
75447 the host where execution occurs. These files (with implementation-defined names) shall be
75448 retained in the home directory of the user under whose user identifier the batch job is executed.

75449 **Mail_Points and Mail_Users Attributes**

75450 A batch server that executes a batch job for which one of the values of the *Mail_Points* attribute is
75451 the value *MAIL_AT_BEGINNING* shall send a mail message to each user account listed in the
75452 *Mail_Users* attribute of the batch job.

75453 The mail message shall contain at least the batch job identifier, queue, and server at which the
75454 batch job currently resides, and the *Job_Owner* attribute.

75455 **Output_Path Attribute**

75456 The *Output_Path* attribute of a running job cannot be changed by a *Modify Batch Job Request*.
75457 When the *Keep_Files* attribute of the batch job does not contain the value *KEEP_STD_OUTPUT*, a
75458 batch server that executes a batch job shall either:

- 75459 • Set the standard output stream of the session leader of the batch job to the destination
75460 described by the value of the *Output_Path* attribute of the batch job.

75461 or:

- 75462 • Buffer the standard output of the session leader of the batch job until completion of the
75463 batch job, and when the batch job exits return the contents to the destination described by
75464 the value of the *Output_Path* attribute of the batch job.

75465 When the *Output_Path* attribute does not specify a host name, then the batch server shall retain
75466 the standard output of the batch job on the host of execution.

75467 When the *Keep_Files* attribute does not contain the value *KEEP_STD_OUTPUT* and the
75468 *Output_Path* attribute does specify a host name, then the final destination of the standard output
75469 of the batch job shall be on the host specified.

75470 If the path specified in the *Output_Path* attribute of the batch job is a relative path, the batch
75471 server shall expand the path relative to the home directory of the user on the host to which the
75472 file is being returned.

75473 Whether or not the batch server buffers the standard output of the batch job until completion of
75474 the batch job is implementation-defined. Applications shall not rely on having access to the
75475 standard output of a batch job prior to the completion of the batch job.

75476 When the batch server does buffer the standard output of the batch job and the file cannot be
75477 opened for write upon completion of the batch job, then the batch server shall place the standard
75478 output in an implementation-defined location and notify the user of the location via mail. It shall
75479 be possible for the user to process this mail using the *mailx* utility.

75480 If a batch server that does not buffer the standard output cannot open the standard output path
75481 of the batch job for write access, then the batch server shall abort the batch job.

Priority Attribute

A batch server implementation may choose to preferentially execute a batch job based on the *Priority* attribute. The interpretation of the batch job *Priority* attribute by a batch server is implementation-defined. If an implementation uses the *Priority* attribute, it shall interpret larger values of the *Priority* attribute to mean the batch job shall be preferentially selected for execution.

Rerunable Attribute

A batch job that began execution but did not complete, because the batch server either shut down or terminated abnormally, shall be requeued if the *Rerunable* attribute of the batch job has the value TRUE.

If a batch job, which was requeued after beginning execution but prior to completion, has a valid checkpoint file and the batch server supports checkpointing, then the batch job shall be restarted from the last valid checkpoint.

If the batch job cannot be restarted from a checkpoint, then when a batch job has a *Rerunable* attribute value of TRUE and was requeued after beginning execution but prior to completion, the batch server shall place the batch job into execution at the beginning of the job.

When a batch job has a *Rerunable* attribute value other than TRUE and was requeued after beginning execution but prior to completion, and the batch job cannot be restarted from a checkpoint, then the batch server shall abort the batch job.

Resource_List Attribute

A batch server that executes a batch job shall establish the resource limits of the session leader of the batch job according to the values of the *Resource_List* attribute of the batch job. Resource limits shall be enforced by an implementation-defined method.

Shell_Path_List Attribute

The *Shell_Path_List* job attribute consists of a list of pairs of pathname and host name values. The host name component can be omitted, in which case the pathname serves as the default pathname when a batch server cannot find the name of the host on which it is running in the list.

A batch server that executes a batch job shall select, from the value of the *Shell_Path_List* attribute of the batch job, a pathname where the shell to execute the batch job shall be found. The batch server shall select the pathname, in order of preference, according to the following methods:

- Select the pathname that contains the name of the host on which the batch server is running.
- Select the pathname for which the host name has been omitted.
- Select the pathname for the login shell of the user under which the batch job is to execute.

If the shell path value selected is an invalid pathname, the batch server shall abort the batch job.

If the value of the selected pathname from the *Shell_Path_List* attribute of the batch job represents a partial path, the batch server shall expand the path relative to a path that is implementation-defined.

The batch server that executes the batch job shall execute the program that was selected from the *Shell_Path_List* attribute of the batch job. The batch server shall pass the path to the script of the batch job as the first argument to the shell program.

75523 **User_List Attribute**

75524 The *User_List* job attribute consists of a list of pairs of user name and host name values. The host
 75525 name component can be omitted, in which case the user name serves as a default when a batch
 75526 server cannot find the name of the host on which it is running in the list.

75527 A batch server that executes a batch job shall select, from the value of the *User_List* attribute of
 75528 the batch job, a user name under which to create the session leader. The server shall select the
 75529 user name, in order of preference, according to the following methods:

- 75530 • Select the user name of a value that contains the name of the host on which the batch
 75531 server executes.
- 75532 • Select the user name of a value for which the host name has been omitted.
- 75533 • Select the user name from the *Job_Owner* attribute of the batch job.

75534 **Variable_List Attribute**

75535 A batch server that executes a batch job shall create, in the environment of the session leader of
 75536 the batch job, each environment variable listed in the *Variable_List* attribute of the batch job, and
 75537 set the value of each such environment variable to that of the corresponding variable in the
 75538 variable list.

75539 3.2.2.2 *Batch Job Routing*

75540 To route a batch job is to select a queue from a list and move the batch job to that queue.

75541 A batch server that has routing queues, which have been started, shall route the jobs in the
 75542 routing queues owned by the batch server. A batch server may delay the routing of a batch job.
 75543 The algorithm for selecting a batch job and the queue to which it will be routed is
 75544 implementation-defined.

75545 When a routing queue has multiple possible destinations specified, then the precedence of the
 75546 destinations is implementation-defined.

75547 A batch server that routes a batch job to a queue at another server shall move the batch job into
 75548 the target queue with a *Queue Batch Job Request*.

75549 If the target server rejects the *Queue Batch Job Request*, the routing server shall retry routing the
 75550 batch job or abort the batch job. A batch server that retries failed routings shall provide a means
 75551 for the batch administrator to specify the number of retries and the minimum period of time
 75552 between retries. The means by which an administrator specifies the number of retries and the
 75553 delay between retries is implementation-defined. When the number of retries specified by the
 75554 batch administrator has been exhausted, the batch server shall abort the batch job and perform
 75555 the functions of *Batch Job Exit*; see [Section 3.2.2.3](#).

75556 3.2.2.3 *Batch Job Exit*

75557 For each job in the EXITING state, the batch server that exited the batch job shall perform the
 75558 following deferred services in the order specified:

- 75559 1. If buffering standard error, move that file into the location specified by the *Error_Path*
 75560 attribute of the batch job.
- 75561 2. If buffering standard output, move that file into the location specified by the *Output_Path*
 75562 attribute of the batch job.

Batch Environment Services

Batch Services

3. If the *Mail_Points* attribute of the batch job includes MAIL_AT_EXIT, send mail to the users listed in the *Mail_Users* attribute of the batch job. The mail message shall contain at least the batch job identifier, queue, and server at which the batch job currently resides, and the *Job_Owner* attribute.

4. Remove the batch job from the queue.

If a batch server that buffers the standard error output cannot return the standard error file to the standard error path at the time the batch job exits, the batch server shall do one of the following:

- Mail the standard error file to the batch job owner.
- Save the standard error file and mail the location and name of the file where the standard error is stored to the batch job owner.
- Save the standard error file and notify the user by other implementation-defined means.

If a batch server that buffers the standard output cannot return the standard output file to the standard output path at the time the batch job exits, the batch server shall do one of the following:

- Mail the standard output file to the batch job owner.
- Save the standard output file and mail the location and name of the file where the standard output is stored to the batch job owner.
- Save the standard output file and notify the user by other implementation-defined means.

At the conclusion of job exit processing, the batch job is no longer managed by a batch server.

3.2.2.4 Batch Server Restart

A batch server that has been either shutdown or terminated abnormally, and has returned to operation, is said to have “restarted”.

Upon restarting, a batch server shall requeue those jobs managed by the batch server that were in the RUNNING state at the time the batch server shut down and for which the *Rerunable* attribute of the batch job has the value TRUE.

Queues are defined to be non-volatile. A batch server shall store the content of queues that it controls in such a way that server and system shutdowns do not erase the content of the queues.

3.2.2.5 Batch Job Abort

A batch server that cannot perform a deferred service for a batch job shall abort the batch job.

A batch server that aborts a batch job shall perform the following services:

- Delete the batch job from the queue in which it resides.
- If the *Mail_Points* attribute of the batch job includes the value MAIL_AT_ABORT, send mail to the users listed in the value of the *Mail_Users* attribute of the job. The mail message shall contain at least the batch job identifier, queue, and server at which the batch job currently resides, the *Job_Owner* attribute, and the reason for the abort.
- If the batch job was in the RUNNING state, terminate the session leader of the executing job by sending the session leader a SIGKILL, place the batch job in the EXITING state, and perform the actions of *Batch Job Exit*.

3.2.3 Requested Batch Services

This section describes the services provided by batch servers in response to requests from clients. Table 3-5 summarizes the current set of batch service requests and for each gives its type (deferred or not) and whether it is an optional function.

Table 3-5 Batch Services Summary

Batch Service	Deferred	Optional
Batch Job Execution	Yes	No
Batch Job Routing	Yes	No
Batch Job Exit	Yes	No
Batch Server Restart	Yes	No
Batch Job Abort	Yes	No
Delete Batch Job Request	No	No
Hold Batch Job Request	No	No
Batch Job Message Request	No	Yes
Batch Job Status Request	No	No
Locate Batch Job Request	No	Yes
Modify Batch Job Request	No	No
Move Batch Job Request	No	No
Queue Batch Job Request	No	No
Batch Queue Status Request	No	No
Release Batch Job Request	No	No
Rerun Batch Job Request	No	No
Select Batch Jobs Request	No	No
Server Shutdown Request	No	No
Server Status Request	No	No
Signal Batch Job Request	No	No
Track Batch Job Request	No	Yes

If a request is rejected because the batch client is not authorized to perform the action, the batch server shall return the same status as when the batch job does not exist.

3.2.3.1 Delete Batch Job Request

A batch job is defined to have been deleted when it has been removed from the queue in which it resides and not instantiated in another queue. A client requests that the server that manages a batch job delete the batch job. Such a request is called a *Delete Batch Job Request*.

A batch server shall reject a *Delete Batch Job Request* if any of the following statements are true:

- The user of the batch client is not authorized to delete the designated job.
- The designated job is not managed by the batch server.
- The designated job is in a state inconsistent with the delete request.

A batch server may reject a *Delete Batch Job Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server requested to delete a batch job shall delete the batch job if the batch job exists and is not in the EXITING state.

A batch server that deletes a batch job in the RUNNING state shall send a SIGKILL signal to the

*Batch Environment Services**Batch Services*

75645 session leader of the batch job. It is implementation-defined whether additional signals are sent
 75646 to the session leader of the job prior to sending the SIGKILL signal.

75647 A batch server that deletes a batch job in the RUNNING state shall place the batch job in the
 75648 EXITING state after it has killed the session leader of the batch job and shall perform the actions
 75649 of *Batch Job Exit*.

75650 3.2.3.2 *Hold Batch Job Request*

75651 A batch client can request that the batch server add one or more holds to a batch job. Such a
 75652 request is called a *Hold Batch Job Request*.

75653 A batch server shall reject a *Hold Batch Job Request* if any of the following statements are true:

- 75654 • The batch server does not support one or more of the requested holds to be added to the
 75655 batch job.
- 75656 • The user of the batch client is not authorized to add one or more of the requested holds to
 75657 the batch job.
- 75658 • The batch server does not manage the specified job.
- 75659 • The designated job is in the EXITING state.

75660 A batch server may reject a *Hold Batch Job Request* for other implementation-defined reasons. The
 75661 method used to determine whether the user of a client is authorized to perform the requested
 75662 action is implementation-defined.

75663 A batch server that accepts a *Hold Batch Job Request* for a batch job in the RUNNING state shall
 75664 place a hold on the batch job. The effects, if any, the hold will have on a batch job in the
 75665 RUNNING state are implementation-defined.

75666 A batch server that accepts a *Hold Batch Job Request* shall add each type of hold listed in the *Hold*
 75667 *Batch Job Request*, that is not already present, to the value of the *Hold_Types* attribute of the batch
 75668 job.

75669 3.2.3.3 *Batch Job Message Request*

75670 *Batch Job Message Request* is an optional feature of batch servers. If an implementation supports
 75671 *Batch Job Message Request*, the statements in this section apply and the configuration variable
 75672 POSIX2_PBS_MESSAGE shall be set to 1.

75673 A batch client can request that a batch server write a message into certain output files of a batch
 75674 job. Such a request is called a *Batch Job Message Request*.

75675 A batch server shall reject a *Batch Job Message Request* if any of the following statements are true:

- 75676 • The batch server does not support sending messages to jobs.
- 75677 • The user of the batch client is not authorized to post a message to the designated job.
- 75678 • The designated job does not exist on the batch server.
- 75679 • The designated job is not in the RUNNING state.

75680 A batch server may reject a *Batch Job Message Request* for other implementation-defined reasons.
 75681 The method used to determine whether the user of a client is authorized to perform the
 75682 requested action is implementation-defined.

75683 A batch server that accepts a *Batch Job Message Request* shall write the message sent by the batch
 75684 client into the files indicated by the batch client.

Batch Services

Batch Environment Services

75685 3.2.3.4 Batch Job Status Request

75686 A batch client can request that a batch server respond with the status and attributes of a batch
75687 job. Such a request is called a *Batch Job Status Request*.

75688 A batch server shall reject a *Batch Job Status Request* if any of the following statements are true:

- 75689 • The user of the batch client is not authorized to query the status of the designated job.
- 75690 • The designated job is not managed by the batch server.

75691 A batch server may reject a *Batch Job Status Request* for other implementation-defined reasons.
75692 The method used to determine whether the user of a client is authorized to perform the
75693 requested action is implementation-defined.

75694 A batch server that accepts a *Batch Job Status Request* shall return a *Batch Job Status Message* to the
75695 batch client.

75696 A batch server may return other information in response to a *Batch Job Status Request*.

75697 3.2.3.5 Locate Batch Job Request

75698 *Locate Batch Job Request* is an optional feature of batch servers. If an implementation supports
75699 *Locate Batch Job Request*, the statements in this section apply and the configuration variable
75700 POSIX2_PBS_LOCATE shall be set to 1.

75701 A batch client can ask a batch server to respond with the location of a batch job that was created
75702 by the batch server. Such a request is called a *Locate Batch Job Request*.

75703 A batch server that accepts a *Locate Batch Job Request* shall return a *Batch Job Location Message* to
75704 the batch client.

75705 A batch server may reject a *Locate Batch Job Request* for a batch job that was not created by that
75706 server.

75707 A batch server may reject a *Locate Batch Job Request* for a batch job that is no longer managed by
75708 that server; that is, for a batch job that is not in a queue owned by that server.

75709 A batch server may reject a *Locate Batch Job Request* for other implementation-defined reasons.

75710 3.2.3.6 Modify Batch Job Request

75711 Batch clients modify (alter) the attributes of a batch job by making a request to the server that
75712 manages the batch job. Such a request is called a *Modify Batch Job Request*.

75713 A batch server shall reject a *Modify Batch Job Request* if any of the following statements are true:

- 75714 • The user of the batch client is not authorized to make the requested modification to the
75715 batch job.
- 75716 • The designated job is not managed by the batch server.
- 75717 • The requested modification is inconsistent with the state of the batch job.
- 75718 • An unrecognized resource is requested for a batch job in an execution queue.

75719 A batch server may reject a *Modify Batch Job Request* for other implementation-defined reasons.
75720 The method used to determine whether the user of a client is authorized to perform the
75721 requested action is implementation-defined.

75722 A batch server that accepts a *Modify Batch Job Request* shall modify all the specified attributes of
75723 the batch job. A batch server that rejects a *Modify Batch Job Request* shall modify none of the

*Batch Environment Services**Batch Services*

75724 attributes of the batch job.

75725 If the servicing by a batch server of an otherwise valid request would result in no change, then
75726 the batch server shall indicate successful completion of the request.

75727 3.2.3.7 *Move Batch Job Request*

75728 A batch client can request that a batch server move a batch job to another destination. Such a
75729 request is called a *Move Batch Job Request*.

75730 A batch server shall reject a *Move Batch Job Request* if any of the following statements are true:

- 75731 • The user of the batch client is not authorized to remove the designated job from the queue
75732 in which the batch job resides.
- 75733 • The user of the batch client is not authorized to move the designated job to the destination.
- 75734 • The designated job is not managed by the batch server.
- 75735 • The designated job is in the EXITING state.
- 75736 • The destination is inaccessible.

75737 A batch server can reject a *Move Batch Job Request* for other implementation-defined reasons. The
75738 method used to determine whether the user of a client is authorized to perform the requested
75739 action is implementation-defined.

75740 A batch server that accepts a *Move Batch Job Request* shall perform the following services:

- 75741 • Queue the designated job at the destination.
- 75742 • Remove the designated job from the queue in which the batch job resides.

75743 If the destination resides on another batch server, the batch server shall queue the batch job at
75744 the destination by sending a *Queue Batch Job Request* to the other server. If the *Queue Batch Job*
75745 *Request* fails, the batch server shall reject the *Move Batch Job Request*. If the *Queue Batch Job*
75746 *Request* succeeds, the batch server shall remove the batch job from its queue.

75747 The batch server shall not modify any attributes of the batch job.

75748 3.2.3.8 *Queue Batch Job Request*

75749 A batch queue is controlled by one and only one batch server. A batch server is said to own the
75750 queues that it controls. Batch clients make requests of batch servers to have jobs queued. Such a
75751 request is called a *Queue Batch Job Request*.

75752 A batch server requested to queue a batch job for which the queue is not specified shall select an
75753 implementation-defined queue for the batch job. Such a queue is called the “default queue” of
75754 the batch server. The implementation shall provide the means for a batch administrator to
75755 specify the default queue. The queue, whether specified or defaulted, is called the “target
75756 queue”.

75757 A batch server shall reject a *Queue Batch Job Request* if any of the following statements are true:

- 75758 • The client is not authorized to create a batch job in the target queue.
- 75759 • The request specifies a queue that does not exist on the batch server.
- 75760 • The target queue is an execution queue and the batch server cannot satisfy a resource
75761 requirement of the batch job.

Batch Services

Batch Environment Services

- 75762 • The target queue is an execution queue and an unrecognized resource is requested.
- 75763 • The target queue is an execution queue, the batch server does not support checkpointing,
75764 and the value of the *Checkpoint* attribute of the batch job is not NO_CHECKPOINT.
- 75765 • The job requires access to a user identifier that the batch client is not authorized to access.
- 75766 A batch server may reject a *Queue Batch Job Request* for other implementation-defined reasons.
- 75767 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the
75768 PBS_O_QUEUE value is missing from the value of the *Variable_List* attribute of the batch job
75769 shall add that variable to the list and set the value to the name of the target queue. Once set, no
75770 server shall change the value of PBS_O_QUEUE, even if the batch job is moved to another
75771 queue.
- 75772 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the PBS_JOBID
75773 value is missing from the value of the *Variable_List* attribute shall add that variable to the list and
75774 set the value to the batch job identifier assigned by the server in the format:
- 75775 `sequence_number.server`
- 75776 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the
75777 PBS_JOBNAME value is missing from the value of the *Variable_List* attribute of the batch job
75778 shall add that variable to the list and set the value to the *Job Name* attribute of the batch job.
- 75779 3.2.3.9 *Batch Queue Status Request*
- 75780 A batch client can request that a batch server respond with the status and attributes of a queue.
75781 Such a request is called a *Batch Queue Status Request*.
- 75782 A batch server shall reject a *Batch Queue Status Request* if any of the following statements are
75783 true:
- 75784 • The user of the batch client is not authorized to query the status of the designated queue.
- 75785 • The designated queue does not exist on the batch server.
- 75786 A batch server may reject a *Batch Queue Status Request* for other implementation-defined reasons.
75787 The method used to determine whether the user of a client is authorized to perform the
75788 requested action is implementation-defined.
- 75789 A batch server that accepts a *Batch Queue Status Request* shall return a *Batch Queue Status Reply* to
75790 the batch client.
- 75791 3.2.3.10 *Release Batch Job Request*
- 75792 A batch client can request that the server remove one or more holds from a batch job. Such a
75793 request is called a *Release Batch Job Request*.
- 75794 A batch server shall reject a *Release Batch Job Request* if any of the following statements are true:
- 75795 • The user of the batch client is not authorized to remove one or more of the requested holds
75796 from the batch job.
- 75797 • The batch server does not manage the specified job.
- 75798 A batch server may reject a *Release Batch Job Request* for other implementation-defined reasons.
75799 The method used to determine whether the user of a client is authorized to perform the
75800 requested action is implementation-defined.
- 75801 A batch server that accepts a *Release Batch Job Request* shall remove each type of hold listed in the

*Batch Environment Services**Batch Services*

75802 *Release Batch Job Request*, that is present, from the value of the *Hold_Types* attribute of the batch
75803 job.

75804 3.2.3.11 *Rerun Batch Job Request*

75805 To rerun a batch job is to kill the session leader of the batch job and leave the batch job eligible
75806 for re-execution. A batch client can request that a batch server rerun a batch job. Such a request
75807 is called *Rerun Batch Job Request*.

75808 A batch server shall reject a *Rerun Batch Job Request* if any of the following statements are true:

- 75809 • The user of the batch client is not authorized to rerun the designated job.
- 75810 • The *Rerunable* attribute of the designated job has the value FALSE.
- 75811 • The designated job is not in the RUNNING state.
- 75812 • The batch server does not manage the designated job.

75813 A batch server may reject a *Rerun Batch Job Request* for other implementation-defined reasons.
75814 The method used to determine whether the user of a client is authorized to perform the
75815 requested action is implementation-defined.

75816 A batch server that rejects a *Rerun Batch Job Request* shall in no way modify the execution of the
75817 batch job.

75818 A batch server that accepts a request to rerun a batch job shall perform the following services:

- 75819 • Requeue the batch job in the execution queue in which it was executing.
- 75820 • Send a SIGKILL signal to the process group of the session leader of the batch job.

75821 An implementation may indicate to the batch job owner that the batch job has been rerun.
75822 Whether and how the batch job owner is notified that a batch job is rerun is implementation-
75823 defined.

75824 A batch server that reruns a batch job may send other implementation-defined signals to the
75825 session leader of the batch job prior to sending the SIGKILL signal.

75826 A batch server may preferentially select a rerun job for execution. Whether rerun jobs shall be
75827 selected for execution before other jobs is implementation-defined.

75828 3.2.3.12 *Select Batch Jobs Request*

75829 A batch client can request from a batch server a list of jobs managed by that server that match a
75830 list of selection criteria. Such a request is called a *Select Batch Jobs Request*. All the batch jobs
75831 managed by the batch server that receives the request are candidates for selection.

75832 A batch server that accepts a *Select Batch Jobs Request* shall return a list of zero or more job
75833 identifiers that correspond to jobs that meet the selection criteria.

75834 If the batch client is not authorized to query the status of a batch job, the batch server shall not
75835 select the batch job.

Batch Services

Batch Environment Services

75836 3.2.3.13 *Server Shutdown Request*

75837 A batch server is defined to have shut down when it does not respond to requests from clients
 75838 and does not perform deferred services for jobs. A batch client can request that a batch server
 75839 shut down. Such a request is called a *Server Shutdown Request*.

75840 A batch server shall reject a *Server Shutdown Request* from a client that is not authorized to shut
 75841 down the batch server. The method used to determine whether the user of a client is authorized
 75842 to perform the requested action is implementation-defined.

75843 A batch server may reject a *Server Shutdown Request* for other implementation-defined reasons.
 75844 The reasons for which a *Server Shutdown Request* may be rejected are implementation-defined.

75845 At server shutdown, a batch server shall do, in order of preference, one of the following:

- 75846 • If checkpointing is implemented and the batch job is checkpointable, then checkpoint the
 75847 batch job and requeue it.
- 75848 • If the batch job is rerunnable, then requeue the batch job to be rerun (restarted from the
 75849 beginning).
- 75850 • Abort the batch job.

75851 3.2.3.14 *Server Status Request*

75852 A batch client can request that a batch server respond with the status and attributes of the batch
 75853 server. Such a request is called a *Server Status Request*.

75854 A batch server shall reject a *Server Status Request* if the following statement is true:

- 75855 • The user of the batch client is not authorized to query the status of the designated server.

75856 A batch server may reject a *Server Status Request* for other implementation-defined reasons. The
 75857 method used to determine whether the user of a client is authorized to perform the requested
 75858 action is implementation-defined.

75859 A batch server that accepts a *Server Status Request* shall return a *Server Status Reply* to the batch
 75860 client.

75861 3.2.3.15 *Signal Batch Job Request*

75862 A batch client can request that a batch server signal the session leader of a batch job. Such a
 75863 request is called a *Signal Batch Job Request*.

75864 A batch server shall reject a *Signal Batch Job Request* if any of the following statements are true:

- 75865 • The user of the batch client is not authorized to signal the batch job.
- 75866 • The job is not in the RUNNING state.
- 75867 • The batch server does not manage the designated job.
- 75868 • The requested signal is not supported by the implementation.

75869 A batch server may reject a *Signal Batch Job Request* for other implementation-defined reasons.
 75870 The method used to determine whether the user of a client is authorized to perform the
 75871 requested action is implementation-defined.

75872 A batch server that accepts a request to signal a batch job shall send the signal requested by the
 75873 batch client to the process group of the session leader of the batch job.

*Batch Environment Services**Batch Services*75874 3.2.3.16 *Track Batch Job Request*

75875 *Track Batch Job Request* is an optional feature of batch servers. If an implementation supports
 75876 *Track Batch Job Request*, the statements in this section apply and the configuration variable
 75877 POSIX2_PBS_TRACK shall be set to 1.

75878 *Track Batch Job Request* provides a method for tracking the current location of a batch job. Clients
 75879 may use the tracking information to determine the batch server that should receive a batch
 75880 server request.

75881 If *Track Batch Job Request* is supported by a batch server, then when the batch server queues a
 75882 batch job as a result of a *Queue Batch Job Request*, and the batch server is not the batch server that
 75883 created the batch job, the batch server shall send a *Track Batch Job Request* to the batch server that
 75884 created the job.

75885 If *Track Batch Job Request* is supported by a batch server, then the *Track Batch Job Request* may also
 75886 be sent to other servers as a backup to the primary server. The method by which backup servers
 75887 are specified is implementation-defined.

75888 If *Track Batch Job Request* is supported by a batch server that receives a *Track Batch Job Request*,
 75889 then the batch server shall record the current location of the batch job as contained in the
 75890 request.

75891 **3.3 Common Behavior for Batch Environment Utilities**75892 **3.3.1 Batch Job Identifier**

75893 A utility shall recognize *job_identifiers* of the format:

75894 [sequence_number] [.server_name] [@server]

75895 where:

75896 *sequence_number* An integer that, when combined with *server_name*, provides a batch job
 75897 identifier that is unique within the batch system.

75898 *server_name* The name of the batch server to which the batch job was originally submitted.

75899 *server* The name of the batch server that is currently managing the batch job.

75900 If the application omits the batch *server_name* portion of a batch job identifier, a utility shall use
 75901 the name of a default batch server.

75902 If the application omits the batch *server* portion of a batch job identifier, a utility shall use:

- 75903 • The batch server indicated by *server_name*, if present
- 75904 • The name of the default batch server
- 75905 • The name of the batch server that is currently managing the batch job

75906 If only @*server* is specified, then the status of all jobs owned by the user on the requested server
 75907 is listed.

75908 The means by which a utility determines the default batch server is implementation-defined.

75909 If the application presents the batch *server* portion of a batch job identifier to a utility, the utility
 75910 shall send the request to the specified server.

75911 A strictly conforming application shall use the syntax described for the job identifier. Whenever
 75912 a batch job identifier is specified whose syntax is not recognized by an implementation, then a
 75913 message for each error that occurs shall be written to standard error and the utility shall exit
 75914 with an exit status greater than zero.

75915 When a batch job identifier is supplied as an argument to a batch utility and the *server_name*
 75916 portion of the batch job identifier is omitted, then the utility shall use the name of the default
 75917 batch server.

75918 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*
 75919 portion of the batch job identifier is omitted, then the utility shall use either:

- 75920 • The name of the default batch server
- 75921 or:
- 75922 • The name of the batch server that is currently managing the batch job

75923 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*
 75924 portion of the batch job identifier is specified, then the utility shall send the required *Batch Server*
 75925 *Request* to the specified server.

75926 3.3.2 Destination

75927 The utility shall recognize a *destination* of the format:

75928 [queue] [@server]

75929 where:

75930 *queue* The name of a valid execution or routing queue at the batch server denoted by
 75931 @*server*, defined as a string of up to 15 alphanumeric characters in the portable
 75932 character set (see XBD Section 6.1, on page 125) where the first character is
 75933 alphabetic.

75934 *server* The name of a batch server, defined as a string of alphanumeric characters in the
 75935 portable character set.

75936 If the application omits the batch *server* portion of a destination, then the utility shall use either:

- 75937 • The name of the default batch server
- 75938 or:
- 75939 • The name of the batch server that is currently managing the batch job

75940 The means by which a utility determines the default batch server is implementation-defined.

75941 If the application omits the *queue* portion of a destination, then the utility shall use the name of
 75942 the default queue at the batch server chosen. The means by which a batch server determines its
 75943 default queue is implementation-defined. If a destination is specified in the *queue@server* form,
 75944 then the utility shall use the specified queue at the specified server.

75945 A strictly conforming application shall use the syntax described for a destination. Whenever a
 75946 destination is specified whose syntax is not recognized by an implementation, then a message
 75947 shall be written to standard error and the utility shall exit with an exit status greater than zero.

3.3.3 Multiple Keyword-Value Pairs

For each option that can have multiple keyword-value pair arguments, the following rules shall apply. Examples of options that can have list-oriented option-arguments are `-u value@keyword` and `-l keyword=value`.

1. If a batch utility is presented with a list-oriented option-argument for which a keyword has a corresponding value that begins with a single or double-quote, then the utility shall stop interpreting the input stream for delimiters until a second single or double-quote, respectively, is encountered. This feature allows some flexibility for a <comma> (',') or <equals-sign> ('=') to be part of the value string for a particular keyword; for example:

```
keywd1='val1, val2', keywd2="val3, val4"
```

Note: This may require the user to escape the quotes as in the following command:

```
foo -xkeywd1=\'val1, val2\', keywd2=\"val3, val4\"
```

2. If a batch server is presented with a list-oriented attribute that has a keyword that was encountered earlier in the list, then the later entry for that keyword shall replace the earlier entry.
3. If a batch server is presented with a list-oriented attribute that has a keyword without any corresponding value of the form `keyword=` or `@keyword` and the same keyword was encountered earlier in the list, then the prior entry for that keyword shall be ignored by the batch server.
4. If a batch utility is expecting a list-oriented option-argument entry of the form `keyword=value`, but is presented with an entry of the form `keyword` without any corresponding `value`, then the entry shall be treated as though a default value of NULL was assigned (that is, `keyword=NULL`) for entry parsing purposes. The utility shall include only the keyword, not the NULL value, in the associated job attribute.
5. If a batch utility is expecting a list-oriented option-argument entry of the form `value@keyword`, but is presented with an entry of the form `value` without any corresponding `keyword`, then the entry shall be treated as though a keyword of NULL was assigned (that is, `value@NULL`) for entry parsing purposes. The utility shall include only the value, not the NULL keyword, in the associated job attribute.
6. A batch server shall accept a list-oriented attribute that has multiple occurrences of the same keyword, interpreting the keywords, in order, with the last value encountered taking precedence over prior instances of the same keyword. This rule allows, but does not require, a batch utility to preprocess the attribute to remove duplicate keywords.
7. If a batch utility is presented with multiple list-oriented option-arguments on the command line or in script directives, or both, for a single option, then the utility shall concatenate, in order, any command line keyword and value pairs to the end of any directive keyword and value pairs separated by a single <comma> to produce a single string that is an equivalent, valid option-argument. The resulting string shall be assigned to the associated attribute of the batch job (after optionally removing duplicate entries as described in item 6).

(blank page)

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

75988

Chapter 4

75989

Utilities

75990

This chapter contains the definitions of the utilities, as follows:

75991

- Mandatory utilities that are present on every conformant system

75992

- Optional utilities that are present only on systems supporting the associated option; see [Section 1.7.1](#) (on page 7) for information on the options in this volume of POSIX.1-2008

75993

admin

Utilities

75994 **NAME**75995 **admin** — create and administer SCCS files (**DEVELOPMENT**)75996 **SYNOPSIS**

```

75997 xSI  admin -i[name] [-n] [-a login] [-d flag] [-e login] [-f flag]
75998         [-m mrlist] [-r rel] [-t[name] [-y[comment]]] newfile
75999
76000 admin -n [-a login] [-d flag] [-e login] [-f flag] [-m mrlist]
76001         [-t[name]] [-y[comment]] newfile...
76002
76003 admin [-a login] [-d flag] [-m mrlist] [-r rel] [-t[name]] file...
76004
76005 admin -h file...
76006
76007 admin -z file...

```

76004 **DESCRIPTION**

76005 The *admin* utility shall create new SCCS files or change parameters of existing ones. If a named
 76006 file does not exist, it shall be created, and its parameters shall be initialized according to the
 76007 specified options. Parameters not initialized by an option shall be assigned a default value. If a
 76008 named file does exist, parameters corresponding to specified options shall be changed, and other
 76009 parameters shall be left as is.

76010 All SCCS filenames supplied by the application shall be of the form *s.filename*. New SCCS files
 76011 shall be given read-only permission mode. Write permission in the parent directory is required
 76012 to create a file. All writing done by *admin* shall be to a temporary *x-file*, named *x.filename* (see *get*)
 76013 created with read-only mode if *admin* is creating a new SCCS file, or created with the same mode
 76014 as that of the SCCS file if the file already exists. After successful execution of *admin*, the SCCS file
 76015 shall be removed (if it exists), and the *x-file* shall be renamed with the name of the SCCS file. This
 76016 ensures that changes are made to the SCCS file only if no errors occur.

76017 The *admin* utility shall also use a transient lock file (named *z.filename*), which is used to prevent
 76018 simultaneous updates to the SCCS file; see *get*.

76019 **OPTIONS**

76020 The *admin* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the *-i*, *-t*, and *-y*
 76021 options have optional option-arguments. These optional option-arguments shall not be
 76022 presented as separate arguments. The following options are supported:

- 76023 **-n** Create a new SCCS file. When *-n* is used without *-i*, the SCCS file shall be created
 76024 with control information but without any file data.
- 76025 **-i[name]** Specify the *name* of a file from which the text for a new SCCS file shall be taken.
 76026 The text constitutes the first delta of the file (see the *-r* option for the delta
 76027 numbering scheme). If the *-i* option is used, but the *name* option-argument is
 76028 omitted, the text shall be obtained by reading the standard input. If this option is
 76029 omitted, the SCCS file shall be created with control information but without any
 76030 file data. The *-i* option implies the *-n* option.
- 76031 **-r SID** Specify the SID of the initial delta to be inserted. This SID shall be a trunk SID; that
 76032 is, the branch and sequence numbers shall be zero or missing. The level number is
 76033 optional, and defaults to 1.
- 76034 **-t[name]** Specify the *name* of a file from which descriptive text for the SCCS file shall be
 76035 taken. In the case of existing SCCS files (neither *-i* nor *-n* is specified):

76036		• A -t option without a <i>name</i> option-argument shall cause the removal of
76037		descriptive text (if any) currently in the SCCS file.
76038		• A -t option with a <i>name</i> option-argument shall cause the text (if any) in the
76039		named file to replace the descriptive text (if any) currently in the SCCS file.
76040	-f flag	Specify a <i>flag</i> , and, possibly, a value for the <i>flag</i> , to be placed in the SCCS file.
76041		Several -f options may be supplied on a single <i>admin</i> command line.
76042		Implementations shall recognize the following flags and associated values:
76043	b	Allow use of the -b option on a <i>get</i> command to create branch deltas.
76044	cceil	Specify the highest release (that is, ceiling), a number less than or equal to
76045		9 999, which may be retrieved by a <i>get</i> command for editing. The default
76046		value for an unspecified c flag shall be 9 999.
76047	ffloor	Specify the lowest release (that is, floor), a number greater than 0 but less
76048		than 9 999, which may be retrieved by a <i>get</i> command for editing. The
76049		default value for an unspecified f flag shall be 1.
76050	dSID	Specify the default delta number (SID) to be used by a <i>get</i> command.
76051	istr	Treat the “No ID keywords” message issued by <i>get</i> or <i>delta</i> as a fatal error.
76052		In the absence of this flag, the message is only a warning. The message is
76053		issued if no SCCS identification keywords (see <i>get</i>) are found in the text
76054		retrieved or stored in the SCCS file. If a value is supplied, the application
76055		shall ensure that the keywords exactly match the given string; however,
76056		the string shall contain a keyword, and no embedded <newline>
76057		characters.
76058	j	Allow concurrent <i>get</i> commands for editing on the same SID of an SCCS
76059		file. This allows multiple concurrent updates to the same version of the
76060		SCCS file.
76061	llist	Specify a <i>list</i> of releases to which deltas can no longer be made (that is, <i>get</i>
76062		-e against one of these locked releases fails). Conforming applications
76063		shall use the following syntax to specify a <i>list</i> . Implementations may
76064		accept additional forms as an extension:
76065		<list> ::= a <range-list>
76066		<range-list> ::= <range> <range-list>, <range>
76067		<range> ::= <SID>
76068		The character <i>a</i> in the <i>list</i> shall be equivalent to specifying all releases for
76069		the named SCCS file. The non-terminal <SID> in range shall be the delta
76070		number of an existing delta associated with the SCCS file.
76071	n	Cause <i>delta</i> to create a null delta in each of those releases (if any) being
76072		skipped when a delta is made in a new release (for example, in making
76073		delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas
76074		shall serve as anchor points so that branch deltas may later be created
76075		from them. The absence of this flag shall cause skipped releases to be
76076		nonexistent in the SCCS file, preventing branch deltas from being created
76077		from them in the future. During the initial creation of an SCCS file, the n
76078		flag may be ignored; that is, if the -r option is used to set the release
76079		number of the initial SID to a value greater than 1, null deltas need not be
76080		created for the “skipped” releases.

admin

Utilities

76081	qtext	Substitute user-definable <i>text</i> for all occurrences of the %Q% keyword in the SCCS file text retrieved by <i>get</i> .
76082		
76083	mmod	Specify the module name of the SCCS file substituted for all occurrences of the %M% keyword in the SCCS file text retrieved by <i>get</i> . If the m flag is not specified, the value assigned shall be the name of the SCCS file with the leading ' .' removed.
76084		
76085		
76086		
76087	ttype	Specify the <i>type</i> of module in the SCCS file substituted for all occurrences of the %Y% keyword in the SCCS file text retrieved by <i>get</i> .
76088		
76089	vpgm	Cause <i>delta</i> to prompt for modification request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validation program. (If this flag is set when creating an SCCS file, the application shall ensure that the m option is also used even if its value is null.)
76090		
76091		
76092		
76093		
76094	-d flag	Remove (delete) the specified <i>flag</i> from an SCCS file. Several -d options may be supplied on a single <i>admin</i> command. See the -f option for allowable <i>flag</i> names. (The l list flag gives a <i>list</i> of releases to be unlocked. See the -f option for further description of the l flag and the syntax of a <i>list</i> .)
76095		
76096		
76097		
76098	-a login	Specify a <i>login</i> name, or numerical group ID, to be added to the list of users who may make deltas (changes) to the SCCS file. A group ID shall be equivalent to specifying all <i>login</i> names common to that group ID. Several -a options may be used on a single <i>admin</i> command line. As many <i>logins</i> , or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If <i>login</i> or group ID is preceded by a '!', the users so specified shall be denied permission to make deltas.
76099		
76100		
76101		
76102		
76103		
76104		
76105	-e login	Specify a <i>login</i> name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all <i>login</i> names common to that group ID. Several -e options may be used on a single <i>admin</i> command line.
76106		
76107		
76108		
76109	-y[comment]	Insert the <i>comment</i> text into the SCCS file as a comment for the initial delta in a manner identical to that of <i>delta</i> . In the POSIX locale, omission of the -y option shall result in a default comment line being inserted in the form:
76110		
76111		
76112		"date and time created %s %s by %s", <date>, <time>, <login>
76113		where <date> is expressed in the format of the <i>date</i> utility's %y/%m/%d conversion specification, <time> in the format of the <i>date</i> utility's %T conversion specification format, and <login> is the login name of the user creating the file.
76114		
76115		
76116	-m mrlist	Insert the list of modification request (MR) numbers into the SCCS file as the reason for creating the initial delta in a manner identical to <i>delta</i> . The application shall ensure that the v flag is set and the MR numbers are validated if the v flag has a value (the name of an MR number validation program). A diagnostic message shall be written if the v flag is not set or MR validation fails.
76117		
76118		
76119		
76120		
76121	-h	Check the structure of the SCCS file and compare the newly computed checksum with the checksum that is stored in the SCCS file. If the newly computed checksum does not match the checksum in the SCCS file, a diagnostic message shall be written.
76122		
76123		
76124		

Utilities

admin

76125 **-z** Recompute the SCCS file checksum and store it in the first line of the SCCS file (see
 76126 the **-h** option above). Note that use of this option on a truly corrupted file may
 76127 prevent future detection of the corruption.

76128 **OPERANDS**

76129 The following operands shall be supported:

76130 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *admin*
 76131 utility shall behave as though each file in the directory were specified as a named
 76132 file, except that non-SCCS files (last component of the pathname does not begin
 76133 with **s.**) and unreadable files shall be silently ignored.

76134 *newfile* A pathname of an SCCS file to be created.

76135 If exactly one *file* or *newfile* operand appears, and it is **'-'**, the standard input shall be read; each
 76136 line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-
 76137 SCCS files and unreadable files shall be silently ignored.

76138 **STDIN**

76139 The standard input shall be a text file used only if **-i** is specified without an option-argument or
 76140 if a *file* or *newfile* operand is specified as **'-'**. If the first character of any standard input line is
 76141 <SOH> in the POSIX locale, the results are unspecified.

76142 **INPUT FILES**

76143 The existing SCCS files shall be text files of an unspecified format.

76144 The application shall ensure that the file named by the **-i** option's *name* option-argument shall
 76145 be a text file; if the first character of any line in this file is <SOH> in the POSIX locale, the results
 76146 are unspecified. If this file contains more than 99 999 lines, the number of lines recorded in the
 76147 header for this file shall be 99 999 for this delta.

76148 **ENVIRONMENT VARIABLES**

76149 The following environment variables shall affect the execution of *admin*:

76150 *LANG* Provide a default value for the internationalization variables that are unset or null.
 76151 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 76152 variables used to determine the values of locale categories.)

76153 *LC_ALL* If set to a non-empty string value, override the values of all the other
 76154 internationalization variables.

76155 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 76156 characters (for example, single-byte as opposed to multi-byte characters in
 76157 arguments and input files).

76158 *LC_MESSAGES*

76159 Determine the locale that should be used to affect the format and contents of
 76160 diagnostic messages written to standard error and the contents of the default **-y**
 76161 comment.

76162 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

76163 **ASYNCHRONOUS EVENTS**

76164 Default.

76165 **STDOUT**

76166 Not used.

admin*Utilities*76167 **STDERR**

76168 The standard error shall be used only for diagnostic messages.

76169 **OUTPUT FILES**

76170 Any SCCS files created shall be text files of an unspecified format. During processing of a *file*, a
 76171 locking *z-file*, as described in *get* (on page 2764), may be created and deleted.

76172 **EXTENDED DESCRIPTION**

76173 None.

76174 **EXIT STATUS**

76175 The following exit values shall be returned:

76176 0 Successful completion.

76177 >0 An error occurred.

76178 **CONSEQUENCES OF ERRORS**

76179 Default.

76180 **APPLICATION USAGE**

76181 It is recommended that directories containing SCCS files be writable by the owner only, and that
 76182 SCCS files themselves be read-only. The mode of the directories should allow only the owner to
 76183 modify SCCS files contained in the directories. The mode of the SCCS files prevents any
 76184 modification at all except by SCCS commands.

76185 **EXAMPLES**

76186 None.

76187 **RATIONALE**

76188 None.

76189 **FUTURE DIRECTIONS**

76190 None.

76191 **SEE ALSO**

76192 *delta*, *get*, *prs*, *what*

76193 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

76194 **CHANGE HISTORY**

76195 First released in Issue 2.

76196 **Issue 6**

76197 The normative text is reworded to avoid use of the term “must” for application requirements,
 76198 and to emphasize the term “shall” for implementation requirements.

76199 The grammar is updated.

76200 The Open Group Base Resolution bwg2001-007 is applied, adding new text to the INPUT FILES
 76201 section warning that the maximum lines recorded in the file is 99 999.

76202 The Open Group Base Resolution bwg2001-009 is applied, amending the description of the **-h**
 76203 option.

NAME

alias — define or display aliases

SYNOPSIS

alias [*alias-name*[=*string*] ...]

DESCRIPTION

The *alias* utility shall create or redefine alias definitions or write the values of existing alias definitions to standard output. An alias definition provides a string value that shall replace a command name when it is encountered; see [Section 2.3.1](#) (on page 2300).

An alias definition shall affect the current shell execution environment and the execution environments of the subshells of the current shell. When used as specified by this volume of POSIX.1-2008, the alias definition shall not affect the parent process of the current shell nor any utility environment invoked by the shell; see [Section 2.12](#) (on page 2331).

OPTIONS

None.

OPERANDS

The following operands shall be supported:

alias-name Write the alias definition to standard output.

alias-name=string

Assign the value of *string* to the alias *alias-name*.

If no operands are given, all alias definitions shall be written to standard output.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *alias*:

LANG Provide a default value for the internationalization variables that are unset or null. (See [XBD Section 8.2](#) (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI NLSPATH Determine the location of message catalogs for the processing of **LC_MESSAGES**.

ASYNCHRONOUS EVENTS

Default.

alias

Utilities

76244 **STDOUT**

76245 The format for displaying aliases (when no operands or only *name* operands are specified) shall
76246 be:

76247 "%s=%s\n", *name*, *value*

76248 The *value* string shall be written with appropriate quoting so that it is suitable for reinput to the
76249 shell. See the description of shell quoting in [Section 2.2](#) (on page 2298).

76250 **STDERR**

76251 The standard error shall be used only for diagnostic messages.

76252 **OUTPUT FILES**

76253 None.

76254 **EXTENDED DESCRIPTION**

76255 None.

76256 **EXIT STATUS**

76257 The following exit values shall be returned:

76258 0 Successful completion.

76259 >0 One of the *name* operands specified did not have an alias definition, or an error occurred.

76260 **CONSEQUENCES OF ERRORS**

76261 Default.

76262 **APPLICATION USAGE**

76263 None.

76264 **EXAMPLES**

76265 1. Create a short alias for a commonly used *ls* command:

76266 alias lf="ls -CF"

76267 2. Create a simple “redo” command to repeat previous entries in the command history file:

76268 alias r='fc -s'

76269 3. Use 1K units for *du*:

76270 alias du=du\ -k

76271 4. Set up *nohup* so that it can deal with an argument that is itself an alias name:

76272 alias nohup="nohup "

76273 **RATIONALE**

76274 The *alias* description is based on historical KornShell implementations. Known differences exist
76275 between that and the C shell. The KornShell version was adopted to be consistent with all the
76276 other KornShell features in this volume of POSIX.1-2008, such as command line editing.

76277 Since *alias* affects the current shell execution environment, it is generally provided as a shell
76278 regular built-in.

76279 Historical versions of the KornShell have allowed aliases to be exported to scripts that are
76280 invoked by the same shell. This is triggered by the *alias -x* flag; it is allowed by this volume of
76281 POSIX.1-2008 only when an explicit extension such as *-x* is used. The standard developers
76282 considered that aliases were of use primarily to interactive users and that they should normally
76283 not affect shell scripts called by those users; functions are available to such scripts.

76284 Historical versions of the KornShell had not written aliases in a quoted manner suitable for
 76285 reentry to the shell, but this volume of POSIX.1-2008 has made this a requirement for all similar
 76286 output. Therefore, consistency was chosen over this detail of historical practice.

76287 **FUTURE DIRECTIONS**

76288 None.

76289 **SEE ALSO**

76290 [Section 2.9.5](#) (on page 2324)

76291 XBD [Chapter 8](#) (on page 173)

76292 **CHANGE HISTORY**

76293 First released in Issue 4.

76294 **Issue 6**

76295 This utility is marked as part of the User Portability Utilities option.

76296 The APPLICATION USAGE section is added.

76297 **Issue 7**

76298 The *alias* utility is moved from the User Portability Utilities option to the Base. User Portability
 76299 Utilities is now an option for interactive utilities.

76300 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

76301 The first example is changed to remove the creation of an alias for a standard utility that alters
 76302 its behavior to be non-conforming.

76303 NAME

76304 ar — create and maintain library archives

76305 SYNOPSIS

76306 SD ar -d [-v] *archive file...*76307 XSI ar -m [-v] *archive file...*76308 ar -m -a [-v] *posname archive file...*76309 ar -m -b [-v] *posname archive file...*76310 ar -m -i [-v] *posname archive file...*76311 XSI ar -p [-v] [-s] *archive [file...]*76312 XSI ar -q [-cv] *archive file...*76313 ar -r [-cuv] *archive file...*76314 XSI ar -r -a [-cuv] *posname archive file...*76315 ar -r -b [-cuv] *posname archive file...*76316 ar -r -i [-cuv] *posname archive file...*76317 XSI ar -t [-v] [-s] *archive [file...]*76318 XSI ar -x [-v] [-sCT] *archive [file...]*

76319 DESCRIPTION

76320 The *ar* utility is part of the Software Development Utilities option.

76321 The *ar* utility can be used to create and maintain groups of files combined into an archive. Once
 76322 an archive has been created, new files can be added, and existing files in an archive can be
 76323 extracted, deleted, or replaced. When an archive consists entirely of valid object files, the
 76324 implementation shall format the archive so that it is usable as a library for link editing (see *c99*
 76325 and *fort77*). When some of the archived files are not valid object files, the suitability of the
 76326 XSI archive for library use is undefined. If an archive consists entirely of printable files, the entire
 76327 archive shall be printable.

76328 When *ar* creates an archive, it creates administrative information indicating whether a symbol
 76329 table is present in the archive. When there is at least one object file that *ar* recognizes as such in
 76330 the archive, an archive symbol table shall be created in the archive and maintained by *ar*; it is
 76331 used by the link editor to search the archive. Whenever the *ar* utility is used to create or update
 76332 the contents of such an archive, the symbol table shall be rebuilt. The *-s* option shall force the
 76333 symbol table to be rebuilt.

76334 All *file* operands can be pathnames. However, files within archives shall be named by a filename,
 76335 which is the last component of the pathname used when the file was entered into the archive.
 76336 The comparison of *file* operands to the names of files in archives shall be performed by
 76337 comparing the last component of the operand to the name of the file in the archive.

76338 It is unspecified whether multiple files in the archive may be identically named. In the case of
 76339 XSI such files, however, each *file* and *posname* operand shall match only the first file in the archive
 76340 having a name that is the same as the last component of the operand.

OPTIONS

The *ar* utility shall conform to XBD Section 12.2 (on page 215), except for Guideline 9.

The following options shall be supported:

- a** Position new files in the archive after the file named by the *posname* operand.
- b** Position new files in the archive before the file named by the *posname* operand.
- c** Suppress the diagnostic message that is written to standard error by default when the archive *archive* is created.
- C** Prevent extracted files from replacing like-named files in the file system. This option is useful when **-T** is also used, to prevent truncated filenames from replacing files with the same prefix.
- d** Delete one or more *files* from *archive*.
- i** Position new files in the archive before the file in the archive named by the *posname* operand (equivalent to **-b**).
- m** Move the named files in the archive. The **-a**, **-b** or **-i** options with the *posname* operand indicate the position; otherwise, move the names files in the archive to the end of the archive.
- p** Write the contents of the *files* in the archive named by *file* operands from *archive* to the standard output. If no *file* operands are specified, the contents of all files in the archive shall be written in the order of the archive.
- q** Append the named files to the end of the archive. In this case *ar* does not check whether the added files are already in the archive. This is useful to bypass the searching otherwise done when creating a large archive piece by piece.
- r** Replace or add *files* to *archive*. If the archive named by *archive* does not exist, a new archive shall be created and a diagnostic message shall be written to standard error (unless the **-c** option is specified). If no *files* are specified and the *archive* exists, the results are undefined. Files that replace existing files in the archive shall not change the order of the archive. Files that do not replace existing files in the archive shall be appended to the archive unless a **-a**, **-b**, or **-i** option specifies another position.
- s** Force the regeneration of the archive symbol table even if *ar* is not invoked with an option that modifies the archive contents. This option is useful to restore the archive symbol table after it has been stripped; see *strip*.
- t** Write a table of contents of *archive* to the standard output. Only the files specified by the *file* operands shall be included in the written list. If no *file* operands are specified, all files in *archive* shall be included in the order of the archive.
- T** Allow filename truncation of extracted files whose archive names are longer than the file system can support. By default, extracting a file with a name that is too long shall be an error; a diagnostic message shall be written and the file shall not be extracted.
- u** Update older files in the archive. When used with the **-r** option, files in the archive shall be replaced only if the corresponding *file* has a modification time that is at least as new as the modification time of the file in the archive.

76382	-v	Give verbose output. When used with the option characters -d , -r , or -x , write a detailed file-by-file description of the archive creation and maintenance activity, as described in the STDOUT section.
76383		
76384		
76385		When used with -p , write the name of the file in the archive to the standard output before writing the file in the archive itself to the standard output, as described in the STDOUT section.
76386		
76387		
76388		When used with -t , include a long listing of information about the files in the archive, as described in the STDOUT section.
76389		
76390	-x	Extract the files in the archive named by the <i>file</i> operands from <i>archive</i> . The contents of the archive shall not be changed. If no <i>file</i> operands are given, all files in the archive shall be extracted. The modification time of each file extracted shall be set to the time the file is extracted from the archive.
76391		
76392		
76393		

76394 OPERANDS

76395 The following operands shall be supported:

76396	<i>archive</i>	A pathname of the archive.
76397	<i>file</i>	A pathname. Only the last component shall be used when comparing against the names of files in the archive. If two or more <i>file</i> operands have the same last pathname component (basename), the results are unspecified. The implementation's archive format shall not truncate valid filenames of files added to or replaced in the archive.
76398		
76399		
76400		
76401		

76402	XSI	<i>posname</i>	The name of a file in the archive, used for relative positioning; see options -m and -r .
76403			

76404 STDIN

76405 Not used.

76406 INPUT FILES

76407 The archive named by *archive* shall be a file in the format created by *ar -r*.

76408 ENVIRONMENT VARIABLES

76409 The following environment variables shall affect the execution of *ar*:

76410	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
76411		
76412		
76413	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
76414		
76415	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).
76416		
76417		
76418	<i>LC_MESSAGES</i>	
76419		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
76420		
76421	<i>LC_TIME</i>	Determine the format and content for date and time strings written by <i>ar -tv</i> .
76422	XSI	<i>NLSPATH</i> Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .

Utilities

ar

76423	<i>TMPDIR</i>	Determine the pathname that overrides the default directory for temporary files, if any.
76424		
76425	<i>TZ</i>	Determine the timezone used to calculate date and time strings written by <i>ar -tv</i> . If <i>TZ</i> is unset or null, an unspecified default timezone shall be used.
76426		
76427	ASYNCHRONOUS EVENTS	
76428	Default.	
76429	STDOUT	
76430	If the -d option is used with the -v option, the standard output format shall be:	
76431	<code>"d - %s\n", <file></code>	
76432	where <i>file</i> is the operand specified on the command line.	
76433	If the -p option is used with the -v option, <i>ar</i> shall precede the contents of each file with:	
76434	<code>"\n<%s>\n\n", <file></code>	
76435	where <i>file</i> is the operand specified on the command line, if <i>file</i> operands were specified, and the name of the file in the archive if they were not.	
76436		
76437	If the -r option is used with the -v option:	
76438	• If <i>file</i> is already in the archive, the standard output format shall be:	
76439	<code>"r - %s\n", <file></code>	
76440	where <i><file></i> is the operand specified on the command line.	
76441	• If <i>file</i> is not already in the archive, the standard output format shall be:	
76442	<code>"a - %s\n", <file></code>	
76443	where <i><file></i> is the operand specified on the command line.	
76444	If the -t option is used, <i>ar</i> shall write the names of the files in the archive to the standard output in the format:	
76445		
76446	<code>"%s\n", <file></code>	
76447	where <i>file</i> is the operand specified on the command line, if <i>file</i> operands were specified, or the name of the file in the archive if they were not.	
76448		
76449	If the -t option is used with the -v option, the standard output format shall be:	
76450	<code>"%s %u/%u %u %s %d %d:%d %d %s\n", <member mode>, <user ID>, <group ID>, <number of bytes in member>, <abbreviated month>, <day-of-month>, <hour>, <minute>, <year>, <file></code>	
76451		
76452		
76453		
76454	where:	
76455	<i><file></i>	Shall be the operand specified on the command line, if <i>file</i> operands were specified, or the name of the file in the archive if they were not.
76456		
76457	<i><member mode></i>	
76458		Shall be formatted the same as the <i><file mode></i> string defined in the STDOUT section of <i>ls</i> , except that the first character, the <i><entry type></i> , is not used; the string represents the file mode of the file in the archive at the time it was added to or replaced in the archive.
76459		
76460		
76461		

76462 The following represent the last-modification time of a file when it was most recently added to
 76463 or replaced in the archive:

76464 <abbreviated month>

76465 Equivalent to the format of the %b conversion specification format in *date*.

76466 <day-of-month>

76467 Equivalent to the format of the %e conversion specification format in *date*.

76468 <hour> Equivalent to the format of the %H conversion specification format in *date*.

76469 <minute> Equivalent to the format of the %M conversion specification format in *date*.

76470 <year> Equivalent to the format of the %Y conversion specification format in *date*.

76471 When *LC_TIME* does not specify the POSIX locale, a different format and order of presentation
 76472 of these fields relative to each other may be used in a format appropriate in the specified locale.

76473 If the *-x* option is used with the *-v* option, the standard output format shall be:

76474 "x - %s\n", <file>

76475 where *file* is the operand specified on the command line, if *file* operands were specified, or the
 76476 name of the file in the archive if they were not.

76477 **STDERR**

76478 The standard error shall be used only for diagnostic messages. The diagnostic message about
 76479 creating a new archive when *-c* is not specified shall not modify the exit status.

76480 **OUTPUT FILES**

76481 Archives are files with unspecified formats.

76482 **EXTENDED DESCRIPTION**

76483 None.

76484 **EXIT STATUS**

76485 The following exit values shall be returned:

76486 0 Successful completion.

76487 >0 An error occurred.

76488 **CONSEQUENCES OF ERRORS**

76489 Default.

76490 **APPLICATION USAGE**

76491 None.

76492 **EXAMPLES**

76493 None.

76494 **RATIONALE**

76495 The archive format is not described. It is recognized that there are several known *ar* formats,
 76496 which are not compatible. The *ar* utility is included, however, to allow creation of archives that
 76497 are intended for use only on one machine. The archive is specified as a file, and it can be moved
 76498 as a file. This does allow an archive to be moved from one machine to another machine that uses
 76499 the same implementation of *ar*.

76500 Utilities such as *pax* (and its forebears *tar* and *cpio*) also provide portable "archives". This is a not
 76501 a duplication; the *ar* utility is included to provide an interface primarily for *make* and the
 76502 compilers, based on a historical model.

In historical implementations, the `-q` option (available on XSI-conforming systems) is known to execute quickly because *ar* does not check on whether the added members are already in the archive. This is useful to bypass the searching otherwise done when creating a large archive piece-by-piece. These remarks may but need not remain true for a brand new implementation of this utility; hence, these remarks have been moved into the RATIONALE.

BSD implementations historically required applications to provide the `-s` option whenever the archive was supposed to contain a symbol table. As in this volume of POSIX.1-2008, System V historically creates or updates an archive symbol table whenever an object file is removed from, added to, or updated in the archive.

The OPERANDS section requires what might seem to be true without specifying it: the archive cannot truncate the filenames below {NAME_MAX}. Some historical implementations do so, however, causing unexpected results for the application. Therefore, this volume of POSIX.1-2008 makes the requirement explicit to avoid misunderstandings.

According to the System V documentation, the options `-dmpqrtx` are not required to begin with a <hyphen> ('-'). This volume of POSIX.1-2008 requires that a conforming application use the leading <hyphen>.

The archive format used by the 4.4 BSD implementation is documented in this RATIONALE as an example:

A file created by *ar* begins with the "magic" string "`!<arch>\n". The rest of the archive is made up of objects, each of which is composed of a header for a file, a possible filename, and the file contents. The header is portable between machine architectures, and, if the file contents are printable, the archive is itself printable.`

The header is made up of six ASCII fields, followed by a two-character trailer. The fields are the object name (16 characters), the file last modification time (12 characters), the user and group IDs (each 6 characters), the file mode (8 characters), and the file size (10 characters). All numeric fields are in decimal, except for the file mode, which is in octal.

The modification time is the file `st_mtime` field. The user and group IDs are the file `st_uid` and `st_gid` fields. The file mode is the file `st_mode` field. The file size is the file `st_size` field. The two-byte trailer is the string "``<newline>`".

Only the name field has any provision for overflow. If any filename is more than 16 characters in length or contains an embedded space, the string "`#1/`" followed by the ASCII length of the name is written in the name field. The file size (stored in the archive header) is incremented by the length of the name. The name is then written immediately following the archive header.

Any unused characters in any of these fields are written as <space> characters. If any fields are their particular maximum number of characters in length, there is no separation between the fields.

Objects in the archive are always an even number of bytes long; files that are an odd number of bytes long are padded with a <newline>, although the size in the header does not reflect this.

The *ar* utility description requires that (when all its members are valid object files) *ar* produce an object code library, which the linkage editor can use to extract object modules. If the linkage editor needs a symbol table to permit random access to the archive, *ar* must provide it; however, *ar* does not require a symbol table.

The BSD `-o` option was omitted. It is a rare conforming application that uses *ar* to extract object

code from a library with concern for its modification time, since this can only be of importance to *make*. Hence, since this functionality is not deemed important for applications portability, the modification time of the extracted files is set to the current time.

There is at least one known implementation (for a small computer) that can accommodate only object files for that system, disallowing mixed object and other files. The ability to handle any type of file is not only historical practice for most implementations, but is also a reasonable expectation.

Consideration was given to changing the output format of *ar -tv* to the same format as the output of *ls -l*. This would have made parsing the output of *ar* the same as that of *ls*. This was rejected in part because the current *ar* format is commonly used and changes would break historical usage. Second, *ar* gives the user ID and group ID in numeric format separated by a <slash>. Changing this to be the user name and group name would not be correct if the archive were moved to a machine that contained a different user database. Since *ar* cannot know whether the archive was generated on the same machine, it cannot tell what to report.

The text on the *-ur* option combination is historical practice—since one filename can easily represent two different files (for example, */a/foo* and */b/foo*), it is reasonable to replace the file in the archive even when the modification time in the archive is identical to that in the file system.

FUTURE DIRECTIONS

None.

SEE ALSO

c99, *date*, *fort77*, *pax*, *strip*

XBD Chapter 8 (on page 173), Section 12.2 (on page 215), <unistd.h>, description of {POSIX_NO_TRUNC}

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the Software Development Utilities option.

The STDOUT description is changed for the *-v* option to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term “must” for application requirements.

The TZ entry is added to the ENVIRONMENT VARIABLES section.

IEEE PASC Interpretation 1003.2 #198 is applied, changing the description to consistently use “file” to refer to a file in the file system hierarchy, “archive” to refer to the archive being operated upon by the *ar* utility, and “file in the archive” to refer to a copy of a file that is contained in the archive.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/10 is applied, making corrections to the SYNOPSIS. The change was needed since the *-a*, *-b*, and *-i* options are mutually-exclusive, and *posname* is required if any of these options is specified.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/11 is applied, correcting the description of the two-byte trailer in RATIONALE which had missed out a backquote. The correct trailer is a backquote followed by a <newline>.

76591 **Issue 7**

76592 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not
76593 apply.

76594 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

76595 The description of the `-t` option is changed to say “Only the files specified ...”.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

asa*Utilities*76596 **NAME**

76597 asa — interpret carriage-control characters

76598 **SYNOPSIS**76599 FR asa [*file...*]76600 **DESCRIPTION**76601 The *asa* utility shall write its input files to standard output, mapping carriage-control characters
76602 from the text files to line-printer control sequences in an implementation-defined manner.76603 The first character of every line shall be removed from the input, and the following actions are
76604 performed.

76605 If the character removed is:

76606 <space> The rest of the line is output without change.

76607 0 A <newline> is output, then the rest of the input line.

76608 1 One or more implementation-defined characters that causes an advance to the next
76609 page shall be output, followed by the rest of the input line.76610 + The <newline> of the previous line shall be replaced with one or more
76611 implementation-defined characters that causes printing to return to column
76612 position 1, followed by the rest of the input line. If the '+' is the first character in
76613 the input, it shall be equivalent to <space>.76614 The action of the *asa* utility is unspecified upon encountering any character other than those
76615 listed above as the first character in a line.76616 **OPTIONS**

76617 None.

76618 **OPERANDS**76619 *file* A pathname of a text file used for input. If no *file* operands are specified, the
76620 standard input shall be used.76621 **STDIN**76622 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*
76623 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
76624 the standard input shall not be used. See the INPUT FILES section.76625 **INPUT FILES**

76626 The input files shall be text files.

76627 **ENVIRONMENT VARIABLES**76628 The following environment variables shall affect the execution of *asa*:76629 LANG Provide a default value for the internationalization variables that are unset or null.
76630 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
76631 variables used to determine the values of locale categories.)76632 LC_ALL If set to a non-empty string value, override the values of all the other
76633 internationalization variables.76634 LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as
76635 characters (for example, single-byte as opposed to multi-byte characters in
76636 arguments and input files).

76637	<i>LC_MESSAGES</i>	
76638		Determine the locale that should be used to affect the format and contents of
76639		diagnostic messages written to standard error.
76640	XSI	<i>NLSPATH</i> Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
76641	ASYNCHRONOUS EVENTS	
76642		Default.
76643	STDOUT	
76644		The standard output shall be the text from the input file modified as described in the
76645		DESCRIPTION section.
76646	STDERR	
76647		None.
76648	OUTPUT FILES	
76649		None.
76650	EXTENDED DESCRIPTION	
76651		None.
76652	EXIT STATUS	
76653		The following exit values shall be returned:
76654	0	All input files were output successfully.
76655	>0	An error occurred.
76656	CONSEQUENCES OF ERRORS	
76657		Default.
76658	APPLICATION USAGE	
76659		None.
76660	EXAMPLES	
76661	1.	The following command:
76662		<i>asa file</i>
76663		permits the viewing of <i>file</i> (created by a program using FORTRAN-style carriage-control
76664		characters) on a terminal.
76665	2.	The following command:
76666		<i>a.out asa lp</i>
76667		formats the FORTRAN output of <i>a.out</i> and directs it to the printer.
76668	RATIONALE	
76669		The <i>asa</i> utility is needed to map “standard” FORTRAN 77 output into a form acceptable to
76670		contemporary printers. Usually, <i>asa</i> is used to pipe data to the <i>lp</i> utility; see <i>lp</i> .
76671		This utility is generally used only by FORTRAN programs. The standard developers decided to
76672		retain <i>asa</i> to avoid breaking the historical large base of FORTRAN applications that put carriage-
76673		control characters in their output files. There is no requirement that a system have a FORTRAN
76674		compiler in order to run applications that need <i>asa</i> .
76675		Historical implementations have used an ASCII <form-feed> in response to a 1 and an ASCII
76676		<carriage-return> in response to a '+' . It is suggested that implementations treat characters
76677		other than 0, 1, and '+' as <space> in the absence of any compelling reason to do otherwise.

76678 However, the action is listed here as “unspecified”, permitting an implementation to provide
76679 extensions to access fast multiple-line slewing and channel seeking in a non-portable manner.

76680 **FUTURE DIRECTIONS**

76681 None.

76682 **SEE ALSO**

76683 *fort77, lp*

76684 XBD *Chapter 8* (on page 173)

76685 **CHANGE HISTORY**

76686 First released in Issue 4.

76687 **Issue 6**

76688 This utility is marked as part of the FORTRAN Runtime Utilities option.

76689 The normative text is reworded to avoid use of the term “must” for application requirements.

76690 **Issue 7**

76691 Austin Group Interpretation 1003.1-2001 #092 is applied.

76692 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

at

NAME

at — execute commands at a later time

SYNOPSIS

at [-m] [-f *file*] [-q *queue*] [-t *time_arg*]

at [-m] [-f *file*] [-q *queue*] *timespec...*

at -r *at_job_id...*

at -l -q *queue*

at -l [*at_job_id...*]

DESCRIPTION

The *at* utility shall read commands from standard input and group them together as an *at-job*, to be executed at a later time.

The *at-job* shall be executed in a separate invocation of the shell, running in a separate process group with no controlling terminal, except that the environment variables, current working directory, file creation mask, and other implementation-defined execution-time attributes in effect when the *at* utility is executed shall be retained and used when the *at-job* is executed.

When the *at-job* is submitted, the *at_job_id* and scheduled time shall be written to standard error. The *at_job_id* is an identifier that shall be a string consisting solely of alphanumeric characters and the <period> character. The *at_job_id* shall be assigned by the system when the job is scheduled such that it uniquely identifies a particular job.

User notification and the processing of the job's standard output and standard error are described under the *-m* option.

XSI

Users shall be permitted to use *at* if their name appears in the file **at.allow** which is located in an implementation-defined directory. If that file does not exist, the file **at.deny**, which is located in an implementation-defined directory, shall be checked to determine whether the user shall be denied access to *at*. If neither file exists, only a process with appropriate privileges shall be allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted. The **at.allow** and **at.deny** files shall consist of one user name per line.

OPTIONS

The *at* utility shall conform to XBD [Section 12.2](#) (on page 215).

The following options shall be supported:

-f file Specify the pathname of a file to be used as the source of the *at-job*, instead of standard input.

-l (The letter ell.) Report all jobs scheduled for the invoking user if no *at_job_id* operands are specified. If *at_job_ids* are specified, report only information for these jobs. The output shall be written to standard output.

-m Send mail to the invoking user after the *at-job* has run, announcing its completion. Standard output and standard error produced by the *at-job* shall be mailed to the user as well, unless redirected elsewhere. Mail shall be sent even if the job produces no output.

If *-m* is not used, the job's standard output and standard error shall be provided to the user by means of mail, unless they are redirected elsewhere; if there is no such output to provide, the implementation need not notify the user of the job's completion.

76736	-q <i>queuenam</i>	
76737		Specify in which queue to schedule a job for submission. When used with the -l
76738		option, limit the search to that particular queue. By default, at-jobs shall be
76739		scheduled in queue <i>a</i> . In contrast, queue <i>b</i> shall be reserved for batch jobs; see
76740		<i>batch</i> . The meanings of all other <i>queuenames</i> are implementation-defined. If -q is
76741		specified along with either of the -t <i>time_arg</i> or <i>timespec</i> arguments, the results are
76742		unspecified.
76743	-r	Remove the jobs with the specified <i>at_job_id</i> operands that were previously
76744		scheduled by the <i>at</i> utility.
76745	-t <i>time_arg</i>	Submit the job to be run at the time specified by the <i>time</i> option-argument, which
76746		the application shall ensure has the format as specified by the <i>touch -t time</i> utility.

76747 OPERANDS

76748 The following operands shall be supported:

76749	<i>at_job_id</i>	The name reported by a previous invocation of the <i>at</i> utility at the time the job was
76750		scheduled.
76751	<i>timespec</i>	Submit the job to be run at the date and time specified. All of the <i>timespec</i> operands
76752		are interpreted as if they were separated by <space> characters and concatenated,
76753		and shall be parsed as described in the grammar at the end of this section. The date
76754		and time shall be interpreted as being in the timezone of the user (as determined
76755		by the <i>TZ</i> variable), unless a timezone name appears as part of <i>time</i> , below.
76756		In the POSIX locale, the following describes the three parts of the time specification
76757		string. All of the values from the <i>LC_TIME</i> categories in the POSIX locale shall be
76758		recognized in a case-insensitive manner.
76759	<i>time</i>	The time can be specified as one, two, or four digits. One-digit and
76760		two-digit numbers shall be taken to be hours; four-digit numbers to
76761		be hours and minutes. The time can alternatively be specified as two
76762		numbers separated by a <colon>, meaning <i>hour:minute</i> . An AM/PM
76763		indication (one of the values from the am_pm keywords in the
76764		<i>LC_TIME</i> locale category) can follow the time; otherwise, a 24-hour
76765		clock time shall be understood. A timezone name can also follow to
76766		further qualify the time. The acceptable timezone names are
76767		implementation-defined, except that they shall be case-insensitive
76768		and the string utc is supported to indicate the time is in Coordinated
76769		Universal Time. In the POSIX locale, the <i>time</i> field can also be one of
76770		the following tokens:
76771	midnight	Indicates the time 12:00 am (00:00).
76772	noon	Indicates the time 12:00 pm.
76773	now	Indicates the current day and time. Invoking <i>at <now></i>
76774		shall submit an at-job for potentially immediate
76775		execution (that is, subject only to unspecified
76776		scheduling delays).
76777	<i>date</i>	An optional <i>date</i> can be specified as either a month name (one of the
76778		values from the mon or abmon keywords in the <i>LC_TIME</i> locale
76779		category) followed by a day number (and possibly year number
76780		preceded by a comma), or a day of the week (one of the values from
76781		the day or abday keywords in the <i>LC_TIME</i> locale category). In the

76782 POSIX locale, two special days shall be recognized:

76783 **today** Indicates the current day.

76784 **tomorrow** Indicates the day following the current day.

76785 If no *date* is given, **today** shall be assumed if the given time is greater
 76786 than the current time, and **tomorrow** shall be assumed if it is less. If
 76787 the given month is less than the current month (and no year is given),
 76788 next year shall be assumed.

76789 *increment* The optional *increment* shall be a number preceded by a <plus-sign>
 76790 ('+') and suffixed by one of the following: **minutes**, **hours**, **days**,
 76791 **weeks**, **months**, or **years**. (The singular forms shall also be accepted.)
 76792 The keyword **next** shall be equivalent to an increment number of +1.
 76793 For example, the following are equivalent commands:

76794 at 2pm + 1 week
 76795 at 2pm next week

76796 The following grammar describes the precise format of *timespec* in the POSIX locale. The general
 76797 conventions for this style of grammar are described in [Section 1.3](#) (on page 2287). This formal
 76798 syntax shall take precedence over the preceding text syntax description. The longest possible
 76799 token or delimiter shall be recognized at a given point. When used in a *timespec*, white space
 76800 shall also delimit tokens.

76801 %token hr24clock_hr_min
 76802 %token hr24clock_hour
 76803 /*
 76804 An hr24clock_hr_min is a one, two, or four-digit number. A one-digit
 76805 or two-digit number constitutes an hr24clock_hour. An hr24clock_hour
 76806 may be any of the single digits [0,9], or may be double digits, ranging
 76807 from [00,23]. If an hr24clock_hr_min is a four-digit number, the
 76808 first two digits shall be a valid hr24clock_hour, while the last two
 76809 represent the number of minutes, from [00,59].
 76810 */

76811 %token wallclock_hr_min
 76812 %token wallclock_hour
 76813 /*
 76814 A wallclock_hr_min is a one, two-digit, or four-digit number.
 76815 A one-digit or two-digit number constitutes a wallclock_hour.
 76816 A wallclock_hour may be any of the single digits [1,9], or may
 76817 be double digits, ranging from [01,12]. If a wallclock_hr_min
 76818 is a four-digit number, the first two digits shall be a valid
 76819 wallclock_hour, while the last two represent the number of
 76820 minutes, from [00,59].
 76821 */

76822 %token minute
 76823 /*
 76824 A minute is a one or two-digit number whose value can be [0,9]
 76825 or [00,59].
 76826 */

76827 %token day_number
 76828 /*

```

76829      A day_number is a number in the range appropriate for the particular
76830      month and year specified by month_name and year_number, respectively.
76831      If no year_number is given, the current year is assumed if the given
76832      date and time are later this year. If no year_number is given and
76833      the date and time have already occurred this year and the month is
76834      not the current month, next year is the assumed year.
76835      */

76836      %token year_number
76837      /*
76838      A year_number is a four-digit number representing the year A.D., in
76839      which the at_job is to be run.
76840      */

76841      %token inc_number
76842      /*
76843      The inc_number is the number of times the succeeding increment
76844      period is to be added to the specified date and time.
76845      */

76846      %token timezone_name
76847      /*
76848      The name of an optional timezone suffix to the time field, in an
76849      implementation-defined format.
76850      */

76851      %token month_name
76852      /*
76853      One of the values from the mon or abmon keywords in the LC_TIME
76854      locale category.
76855      */

76856      %token day_of_week
76857      /*
76858      One of the values from the day or abday keywords in the LC_TIME
76859      locale category.
76860      */

76861      %token am_pm
76862      /*
76863      One of the values from the am_pm keyword in the LC_TIME locale
76864      category.
76865      */

76866      %start timespec
76867      %%
76868      timespec      : time
76869                    | time date
76870                    | time increment
76871                    | time date increment
76872                    | nowspec
76873                    ;
76874      nowspec       : "now"
76875                    | "now" increment
76876                    ;

```

```

76877     time          : hr24clock_hr_min
76878                   | hr24clock_hr_min timezone_name
76879                   | hr24clock_hour ":" minute
76880                   | hr24clock_hour ":" minute timezone_name
76881                   | wallclock_hr_min am_pm
76882                   | wallclock_hr_min am_pm timezone_name
76883                   | wallclock_hour ":" minute am_pm
76884                   | wallclock_hour ":" minute am_pm timezone_name
76885                   | "noon"
76886                   | "midnight"
76887                   ;
76888     date           : month_name day_number
76889                   | month_name day_number "," year_number
76890                   | day_of_week
76891                   | "today"
76892                   | "tomorrow"
76893                   ;
76894     increment      : "+" inc_number inc_period
76895                   | "next" inc_period
76896                   ;
76897     inc_period     : "minute" | "minutes"
76898                   | "hour" | "hours"
76899                   | "day" | "days"
76900                   | "week" | "weeks"
76901                   | "month" | "months"
76902                   | "year" | "years"
76903                   ;

```

STDIN

The standard input shall be a text file consisting of commands acceptable to the shell command language described in [Chapter 2](#) (on page 2297). The standard input shall only be used if no `-f file` option is specified.

INPUT FILES

See the STDIN section.

XSI

The text files `at.allow` and `at.deny`, which are located in an implementation-defined directory, shall contain zero or more user names, one per line, of users who are, respectively, authorized or denied access to the `at` and `batch` utilities.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of `at`:

<code>LANG</code>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
<code>LC_ALL</code>	If set to a non-empty string value, override the values of all the other internationalization variables.
<code>LC_CTYPE</code>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

76923		LC_MESSAGES	
76924			Determine the locale that should be used to affect the format and contents of
76925			diagnostic messages written to standard error and informative messages written to
76926			standard output.
76927	XSI	NLSPATH	Determine the location of message catalogs for the processing of LC_MESSAGES .
76928		LC_TIME	Determine the format and contents for date and time strings written and accepted
76929			by <i>at</i> .
76930		SHELL	Determine a name of a command interpreter to be used to invoke the <i>at</i> job. If the
76931			variable is unset or null, <i>sh</i> shall be used. If it is set to a value other than a name for
76932			<i>sh</i> , the implementation shall do one of the following: use that shell; use <i>sh</i> ; use the
76933			login shell from the user database; or any of the preceding accompanied by a
76934			warning diagnostic about which was chosen.
76935		TZ	Determine the timezone. The job shall be submitted for execution at the time
76936			specified by <i>timespec</i> or <i>-t time</i> relative to the timezone specified by the TZ
76937			variable. If <i>timespec</i> specifies a timezone, it shall override TZ . If <i>timespec</i> does not
76938			specify a timezone and TZ is unset or null, an unspecified default timezone shall
76939			be used.
76940		ASYNCHRONOUS EVENTS	
76941			Default.
76942		STDOUT	
76943			When standard input is a terminal, prompts of unspecified format for each line of the user input
76944			described in the STDIN section may be written to standard output.
76945			In the POSIX locale, the following shall be written to the standard output for each job when jobs
76946			are listed in response to the <i>-l</i> option:
76947			"%s\t%s\n", <i>at_job_id</i> , < <i>date</i> >
76948			where <i>date</i> shall be equivalent in format to the output of:
76949			<i>date</i> +"%a %b %e %T %Y"
76950			The date and time written shall be adjusted so that they appear in the timezone of the user (as
76951			determined by the TZ variable).
76952		STDERR	
76953			In the POSIX locale, the following shall be written to standard error when a job has been
76954			successfully submitted:
76955			"job %s at %s\n", <i>at_job_id</i> , < <i>date</i> >
76956			where <i>date</i> has the same format as that described in the STDOUT section. Neither this, nor
76957			warning messages concerning the selection of the command interpreter, shall be considered a
76958			diagnostic that changes the exit status.
76959			Diagnostic messages, if any, shall be written to standard error.
76960		OUTPUT FILES	
76961			None.
76962		EXTENDED DESCRIPTION	
76963			None.

EXIT STATUS

The following exit values shall be returned:

- 0 The *at* utility successfully submitted, removed, or listed a job or jobs.
- >0 An error occurred.

CONSEQUENCES OF ERRORS

The job shall not be scheduled, removed, or listed.

APPLICATION USAGE

The format of the *at* command line shown here is guaranteed only for the POSIX locale. Other cultures may be supported with substantially different interfaces, although implementations are encouraged to provide comparable levels of functionality.

Since the commands run in a separate shell invocation, running in a separate process group with no controlling terminal, open file descriptors, traps, and priority inherited from the invoking environment are lost.

Some implementations do not allow substitution of different shells using *SHELL*. System V systems, for example, have used the login shell value for the user in */etc/passwd*. To select reliably another command interpreter, the user must include it as part of the script, such as:

```
$ at 1800
myshell myscript
EOT
job ... at ...
$
```

EXAMPLES

1. This sequence can be used at a terminal:

```
at -m 0730 tomorrow
sort < file >outfile
EOT
```

2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a command procedure (the sequence of output redirection specifications is significant):

```
at now + 1 hour <<!
diff file1 file2 2>&1 >outfile | mailx mygroup
!
```

3. To have a job reschedule itself, *at* can be invoked from within the at-job. For example, this daily processing script named **my.daily** runs every day (although *crontab* is a more appropriate vehicle for such work):

```
# my.daily runs every day
daily processing
at now tomorrow < my.daily
```

4. The spacing of the three portions of the POSIX locale *timespec* is quite flexible as long as there are no ambiguities. Examples of various times and operand presentation include:

```
at 0815am Jan 24
at 8 :15amjan24
at now "+ 1day"
at 5 pm FRIday
at '17
```


77008 utc+
77009 30minutes'

77010 RATIONALE

77011 The *at* utility reads from standard input the commands to be executed at a later time. It may be
77012 useful to redirect standard output and standard error within the specified commands.

77013 The *-t time* option was added as a new capability to support an internationalized way of
77014 specifying a time for execution of the submitted job.

77015 Early proposals added a “jobname” concept as a way of giving submitted jobs names that are
77016 meaningful to the user submitting them. The historical, system-specified *at_job_id* gives no
77017 indication of what the job is. Upon further reflection, it was decided that the benefit of this was
77018 not worth the change in historical interface. The *at* functionality is useful in simple
77019 environments, but in large or complex situations, the functionality provided by the Batch
77020 Services option is more suitable.

77021 The *-q* option historically has been an undocumented option, used mainly by the *batch* utility.

77022 The System V *-m* option was added to provide a method for informing users that an at-job had
77023 completed. Otherwise, users are only informed when output to standard error or standard
77024 output are not redirected.

77025 The behavior of *at <now>* was changed in an early proposal from being unspecified to
77026 submitting a job for potentially immediate execution. Historical BSD *at* implementations support
77027 this. Historical System V implementations give an error in that case, but a change to the System
77028 V versions should have no backwards-compatibility ramifications.

77029 On BSD-based systems, a *-u user* option has allowed those with appropriate privileges to access
77030 the work of other users. Since this is primarily a system administration feature and is not
77031 universally implemented, it has been omitted. Similarly, a specification for the output format for
77032 a user with appropriate privileges viewing the queues of other users has been omitted.

77033 The *-f file* option from System V is used instead of the BSD method of using the last operand as
77034 the pathname. The BSD method is ambiguous—does:

77035 *at 1200 friday*

77036 mean the same thing if there is a file named **friday** in the current directory?

77037 The *at_job_id* is composed of a limited character set in historical practice, and it is mandated here
77038 to invalidate systems that might try using characters that require shell quoting or that could not
77039 be easily parsed by shell scripts.

77040 The *at* utility varies between System V and BSD systems in the way timezones are used. On
77041 System V systems, the *TZ* variable affects the at-job submission times and the times displayed
77042 for the user. On BSD systems, *TZ* is not taken into account. The BSD behavior is easily achieved
77043 with the current specification. If the user wishes to have the timezone default to that of the
77044 system, they merely need to issue the *at* command immediately following an unsetting or null
77045 assignment to *TZ*. For example:

77046 *TZ= at noon ...*

77047 gives the desired BSD result.

77048 While the *yacc*-like grammar specified in the OPERANDS section is lexically unambiguous with
77049 respect to the digit strings, a lexical analyzer would probably be written to look for and return
77050 digit strings in those cases. The parser could then check whether the digit string returned is a
77051 valid *day_number*, *year_number*, and so on, based on the context.

77052 **FUTURE DIRECTIONS**

77053 None.

77054 **SEE ALSO**77055 *batch, crontab*

77056 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

77057 **CHANGE HISTORY**

77058 First released in Issue 2.

77059 **Issue 6**

77060 This utility is marked as part of the User Portability Utilities option.

77061 The following new requirements on POSIX implementations derive from alignment with the
77062 Single UNIX Specification:

- 77063 • If **-m** is not used, the job's standard output and standard error are provided to the user by
- 77064 mail.

77065 The effects of using the **-q** and **-t** options as defined in the IEEE P1003.2b draft standard are
77066 specified.

77067 The normative text is reworded to avoid use of the term "must" for application requirements.

77068 **Issue 7**77069 The *at* utility is moved from the User Portability Utilities option to the Base. User Portability
77070 Utilities is now an option for interactive utilities.77071 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced
77072 by the *at* utility.

77073 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

awk*Utilities*77074 **NAME**77075 **awk** — pattern scanning and processing language77076 **SYNOPSIS**77077 **awk** [**-F** *ERE*] [**-v** *assignment*]... *program* [*argument*...]77078 **awk** [**-F** *ERE*] **-f** *progfile* [**-f** *progfile*]... [**-v** *assignment*]...
77079 [*argument*...]77080 **DESCRIPTION**77081 The *awk* utility shall execute programs written in the *awk* programming language, which is
77082 specialized for textual data manipulation. An *awk* program is a sequence of patterns and
77083 corresponding actions. When input is read that matches a pattern, the action associated with that
77084 pattern is carried out.77085 Input shall be interpreted as a sequence of records. By default, a record is a line, less its
77086 terminating <newline>, but this can be changed by using the **RS** built-in variable. Each record of
77087 input shall be matched in turn against each pattern in the program. For each pattern matched,
77088 the associated action shall be executed.77089 The *awk* utility shall interpret each input record as a sequence of fields where, by default, a field
77090 is a string of non-<blank> characters. This default white-space field delimiter can be changed by
77091 using the **FS** built-in variable or **-F** *ERE*. The *awk* utility shall denote the first field in a record \$1,
77092 the second \$2, and so on. The symbol \$0 shall refer to the entire record; setting any other field
77093 causes the re-evaluation of \$0. Assigning to \$0 shall reset the values of all other fields and the **NF**
77094 built-in variable.77095 **OPTIONS**77096 The *awk* utility shall conform to XBD [Section 12.2](#) (on page 215).

77097 The following options shall be supported:

77098 **-F** *ERE* Define the input field separator to be the extended regular expression *ERE*, before
77099 any input is read; see [Regular Expressions](#) (on page 2439).77100 **-f** *progfile* Specify the pathname of the file *progfile* containing an *awk* program. If multiple
77101 instances of this option are specified, the concatenation of the files specified as
77102 *progfile* in the order specified shall be the *awk* program. The *awk* program can
77103 alternatively be specified in the command line as a single argument.77104 **-v** *assignment*77105 The application shall ensure that the *assignment* argument is in the same form as an
77106 *assignment* operand. The specified variable assignment shall occur prior to
77107 executing the *awk* program, including the actions associated with **BEGIN** patterns
77108 (if any). Multiple occurrences of this option can be specified.77109 **OPERANDS**

77110 The following operands shall be supported:

77111 *program* If no **-f** option is specified, the first operand to *awk* shall be the text of the *awk*
77112 program. The application shall supply the *program* operand as a single argument to
77113 *awk*. If the text does not end in a <newline>, *awk* shall interpret the text as if it did.77114 *argument* Either of the following two types of *argument* can be intermixed:77115 *file* A pathname of a file that contains the input to be read, which is
77116 matched against the set of patterns in the program. If no *file* operands
77117 are specified, or if a *file* operand is '-', the standard input shall be
77118 used.

Utilities

awk

assignment An operand that begins with an <underscore> or alphabetic character from the portable character set (see the table in XBD [Section 6.1](#), on page 125), followed by a sequence of underscores, digits, and alphabetic characters from the portable character set, followed by the '=' character, shall specify a variable assignment rather than a pathname. The characters before the '=' represent the name of an *awk* variable; if that name is an *awk* reserved word (see [Grammar](#), on page 2447) the behavior is undefined. The characters following the <equals-sign> shall be interpreted as if they appeared in the *awk* program preceded and followed by a double-quote ('"') character, as a **STRING** token (see [Grammar](#), on page 2447), except that if the last character is an unescaped <backslash>, it shall be interpreted as a literal <backslash> rather than as the first character of the sequence "\". The variable shall be assigned the value of that **STRING** token and, if appropriate, shall be considered a *numeric string* (see [Expressions in awk](#), on page 2433), the variable shall also be assigned its numeric value. Each such variable assignment shall occur just prior to the processing of the following *file*, if any. Thus, an assignment before the first *file* argument shall be executed after the **BEGIN** actions (if any), while an assignment after the last *file* argument shall occur before the **END** actions (if any). If there are no *file* arguments, assignments shall be executed before processing the standard input.

STDIN

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-' ; see the INPUT FILES section. If the *awk* program contains no actions and no patterns, but is otherwise a valid *awk* program, standard input and any *file* operands shall not be read and *awk* shall exit with a return status of zero.

INPUT FILES

Input files to the *awk* program from any of the following sources shall be text files:

- Any *file* operands or their equivalents, achieved by modifying the *awk* variables **ARGV** and **ARGC**
- Standard input in the absence of any *file* operands
- Arguments to the **getline** function

Whether the variable **RS** is set to a value other than a <newline> or not, for these files, implementations shall support records terminated with the specified separator up to {LINE_MAX} bytes and may support longer records.

If **-f progfile** is specified, the application shall ensure that the files named by each of the *progfile* option-arguments are text files and their concatenation, in the same order as they appear in the arguments, is an *awk* program.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *awk*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

77164	LC_ALL	If set to a non-empty string value, override the values of all the other internationalization variables.
77165		
77166	LC_COLLATE	
77167		Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions and in comparisons of string values.
77168		
77169		
77170	LC_CTYPE	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the identification of characters as letters, and the mapping of uppercase and lowercase characters for the toupper and tolower functions.
77171		
77172		
77173		
77174		
77175	LC_MESSAGES	
77176		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
77177		
77178	LC_NUMERIC	
77179		Determine the radix character used when interpreting numeric input, performing conversions between numeric and string values, and formatting numeric output.
77180		Regardless of locale, the <period> character (the decimal-point character of the POSIX locale) is the decimal-point character recognized in processing <i>awk</i> programs (including assignments in command line arguments).
77181		
77182		
77183		
77184	XSI NLSPATH	Determine the location of message catalogs for the processing of LC_MESSAGES .
77185	PATH	Determine the search path when looking for commands executed by <i>system(expr)</i> , or input and output pipes; see XBD Chapter 8 (on page 173).
77186		
77187		In addition, all environment variables shall be visible via the <i>awk</i> variable ENVIRON .
77188	ASYNCHRONOUS EVENTS	
77189		Default.
77190	STDOUT	
77191		The nature of the output files depends on the <i>awk</i> program.
77192	STDERR	
77193		The standard error shall be used only for diagnostic messages.
77194	OUTPUT FILES	
77195		The nature of the output files depends on the <i>awk</i> program.
77196	EXTENDED DESCRIPTION	
77197	Overall Program Structure	
77198		An <i>awk</i> program is composed of pairs of the form:
77199		<i>pattern</i> { <i>action</i> }
77200		Either the pattern or the action (including the enclosing brace characters) can be omitted.
77201		A missing pattern shall match any record of input, and a missing action shall be equivalent to:
77202		{ <i>print</i> }
77203		Execution of the <i>awk</i> program shall start by first executing the actions associated with all BEGIN patterns in the order they occur in the program. Then each <i>file</i> operand (or standard input if no files were specified) shall be processed in turn by reading data from the file until a record
77204		
77205		

separator is seen (<newline> by default). Before the first reference to a field in the record is evaluated, the record shall be split into fields, according to the rules in [Regular Expressions](#) (on page 2439), using the value of **FS** that was current at the time the record was read. Each pattern in the program then shall be evaluated in the order of occurrence, and the action associated with each pattern that matches the current record executed. The action for a matching pattern shall be executed before evaluating subsequent patterns. Finally, the actions associated with all **END** patterns shall be executed in the order they occur in the program.

Expressions in awk

Expressions describe computations used in *patterns* and *actions*. In the following table, valid expression operations are given in groups from highest precedence first to lowest precedence last, with equal-precedence operators grouped between horizontal lines. In expression evaluation, where the grammar is formally ambiguous, higher precedence operators shall be evaluated before lower precedence operators. In this table *expr*, *expr1*, *expr2*, and *expr3* represent any expression, while *lvalue* represents any entity that can be assigned to (that is, on the left side of an assignment operator). The precise syntax of expressions is given in [Grammar](#) (on page 2447).

Table 4-1 Expressions in Decreasing Precedence in *awk*

Syntax	Name	Type of Result	Associativity
(<i>expr</i>)	Grouping	Type of <i>expr</i>	N/A
<i>\$expr</i>	Field reference	String	N/A
<i>lvalue</i> ++	Post-increment	Numeric	N/A
<i>lvalue</i> --	Post-decrement	Numeric	N/A
++ <i>lvalue</i>	Pre-increment	Numeric	N/A
-- <i>lvalue</i>	Pre-decrement	Numeric	N/A
<i>expr</i> ^ <i>expr</i>	Exponentiation	Numeric	Right
! <i>expr</i>	Logical not	Numeric	N/A
+ <i>expr</i>	Unary plus	Numeric	N/A
- <i>expr</i>	Unary minus	Numeric	N/A
<i>expr</i> * <i>expr</i>	Multiplication	Numeric	Left
<i>expr</i> / <i>expr</i>	Division	Numeric	Left
<i>expr</i> % <i>expr</i>	Modulus	Numeric	Left
<i>expr</i> + <i>expr</i>	Addition	Numeric	Left
<i>expr</i> - <i>expr</i>	Subtraction	Numeric	Left
<i>expr</i> <i>expr</i>	String concatenation	String	Left
<i>expr</i> < <i>expr</i>	Less than	Numeric	None
<i>expr</i> <= <i>expr</i>	Less than or equal to	Numeric	None
<i>expr</i> != <i>expr</i>	Not equal to	Numeric	None
<i>expr</i> == <i>expr</i>	Equal to	Numeric	None
<i>expr</i> > <i>expr</i>	Greater than	Numeric	None
<i>expr</i> >= <i>expr</i>	Greater than or equal to	Numeric	None
<i>expr</i> ~ <i>expr</i>	ERE match	Numeric	None
<i>expr</i> !~ <i>expr</i>	ERE non-match	Numeric	None
<i>expr</i> in <i>array</i>	Array membership	Numeric	Left
(<i>index</i>) in <i>array</i>	Multi-dimension array	Numeric	Left

Syntax	Name	Type of Result	Associativity
	membership		
<i>expr</i> && <i>expr</i>	Logical AND	Numeric	Left
<i>expr</i> <i>expr</i>	Logical OR	Numeric	Left
<i>expr1</i> ? <i>expr2</i> : <i>expr3</i>	Conditional expression	Type of selected <i>expr2</i> or <i>expr3</i>	Right
<i>lvalue</i> ^= <i>expr</i>	Exponentiation assignment	Numeric	Right
<i>lvalue</i> %= <i>expr</i>	Modulus assignment	Numeric	Right
<i>lvalue</i> *= <i>expr</i>	Multiplication assignment	Numeric	Right
<i>lvalue</i> /= <i>expr</i>	Division assignment	Numeric	Right
<i>lvalue</i> += <i>expr</i>	Addition assignment	Numeric	Right
<i>lvalue</i> -= <i>expr</i>	Subtraction assignment	Numeric	Right
<i>lvalue</i> = <i>expr</i>	Assignment	Type of <i>expr</i>	Right

Each expression shall have either a string value, a numeric value, or both. Except as stated for specific contexts, the value of an expression shall be implicitly converted to the type needed for the context in which it is used. A string value shall be converted to a numeric value either by the equivalent of the following calls to functions defined by the ISO C standard:

```
setlocale(LC_NUMERIC, "");
numeric_value = atof(string_value);
```

or by converting the initial portion of the string to type **double** representation as follows:

The input string is decomposed into two parts: an initial, possibly empty, sequence of white-space characters (as specified by *isspace()*) and a subject sequence interpreted as a floating-point constant.

The expected form of the subject sequence is an optional '+' or '-' sign, then a non-empty sequence of digits optionally containing a <period>, then an optional exponent part. An exponent part consists of 'e' or 'E', followed by an optional sign, followed by one or more decimal digits.

The sequence starting with the first digit or the <period> (whichever occurs first) is interpreted as a floating constant of the C language, and if neither an exponent part nor a <period> appears, a <period> is assumed to follow the last digit in the string. If the subject sequence begins with a minus-sign, the value resulting from the conversion is negated.

A numeric value that is exactly equal to the value of an integer (see [Section 1.1.2](#), on page 2283) shall be converted to a string by the equivalent of a call to the **sprintf** function (see [String Functions](#), on page 2444) with the string "%d" as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. Any other numeric value shall be converted to a string by the equivalent of a call to the **sprintf** function with the value of the variable **CONVFMT** as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. The result of the conversion is unspecified if the value of **CONVFMT** is not a floating-point format specification. This volume of POSIX.1-2008 specifies no explicit conversions between numbers and strings. An application can force an expression to be treated as a number by adding zero to it, or can force it to be treated as a string by concatenating the null string (" ") to it.

A string value shall be considered a *numeric string* if it comes from one of the following:

1. Field variables

2. Input from the *getline()* function
3. **FILENAME**
4. **ARGV** array elements
5. **ENVIRON** array elements
6. Array elements created by the *split()* function
7. A command line variable assignment
8. Variable assignment from another numeric string variable

and an implementation-dependent condition corresponding to either case (a) or (b) below is met.

- a. After the equivalent of the following calls to functions defined by the ISO C standard, *string_value_end* would differ from *string_value*, and any characters before the terminating null character in *string_value_end* would be <blank> characters:

```
char *string_value_end;
setlocale(LC_NUMERIC, "");
numeric_value = strtod (string_value, &string_value_end);
```

- b. After all the following conversions have been applied, the resulting string would lexically be recognized as a **NUMBER** token as described by the lexical conventions in [Grammar](#) (on page 2447):

- All leading and trailing <blank> characters are discarded.
- If the first non-<blank> is '+' or '-', it is discarded.
- Each occurrence of the decimal point character from the current locale is changed to a <period>.

In case (a) the numeric value of the *numeric string* shall be the value that would be returned by the *strtod()* call. In case (b) if the first non-<blank> is '-', the numeric value of the *numeric string* shall be the negation of the numeric value of the recognized **NUMBER** token; otherwise, the numeric value of the *numeric string* shall be the numeric value of the recognized **NUMBER** token. Whether or not a string is a *numeric string* shall be relevant only in contexts where that term is used in this section.

When an expression is used in a Boolean context, if it has a numeric value, a value of zero shall be treated as false and any other value shall be treated as true. Otherwise, a string value of the null string shall be treated as false and any other value shall be treated as true. A Boolean context shall be one of the following:

- The first subexpression of a conditional expression
- An expression operated on by logical NOT, logical AND, or logical OR
- The second expression of a **for** statement
- The expression of an **if** statement
- The expression of the **while** clause in either a **while** or **do...while** statement
- An expression used as a pattern (as in Overall Program Structure)

All arithmetic shall follow the semantics of floating-point arithmetic as specified by the ISO C standard (see [Section 1.1.2](#), on page 2283).

The value of the expression:

77335 `expr1 ^ expr2`

77336 shall be equivalent to the value returned by the ISO C standard function call:

77337 `pow(expr1, expr2)`

77338 The expression:

77339 `lvalue ^= expr`

77340 shall be equivalent to the ISO C standard expression:

77341 `lvalue = pow(lvalue, expr)`

77342 except that lvalue shall be evaluated only once. The value of the expression:

77343 `expr1 % expr2`

77344 shall be equivalent to the value returned by the ISO C standard function call:

77345 `fmod(expr1, expr2)`

77346 The expression:

77347 `lvalue %= expr`

77348 shall be equivalent to the ISO C standard expression:

77349 `lvalue = fmod(lvalue, expr)`

77350 except that lvalue shall be evaluated only once.

77351 Variables and fields shall be set by the assignment statement:

77352 `lvalue = expression`

77353 and the type of *expression* shall determine the resulting variable type. The assignment includes the arithmetic assignments ("`+=`", "`-=`", "`*=`", "`/=`", "`%=`", "`^=`", "`++`", "`--`") all of which shall produce a numeric result. The left-hand side of an assignment and the target of increment and decrement operators can be one of a variable, an array with index, or a field selector.

77357 The *awk* language supplies arrays that are used for storing numbers or strings. Arrays need not be declared. They shall initially be empty, and their sizes shall change dynamically. The subscripts, or element identifiers, are strings, providing a type of associative array capability. An array name followed by a subscript within square brackets can be used as an lvalue and thus as an expression, as described in the grammar; see [Grammar](#) (on page 2447). Unsubscripted array names can be used in only the following contexts:

- 77363 • A parameter in a function definition or function call
- 77364 • The **NAME** token following any use of the keyword **in** as specified in the grammar (see [Grammar](#), on page 2447); if the name used in this context is not an array name, the behavior is undefined

77367 A valid array *index* shall consist of one or more <comma>-separated expressions, similar to the way in which multi-dimensional arrays are indexed in some programming languages. Because *awk* arrays are really one-dimensional, such a <comma>-separated list shall be converted to a single string by concatenating the string values of the separate expressions, each separated from the other by the value of the **SUBSEP** variable. Thus, the following two index operations shall be equivalent:

77373 `var[expr1, expr2, ... exprn]`

77374 `var[expr1 SUBSEP expr2 SUBSEP ... SUBSEP exprn]`

The application shall ensure that a multi-dimensioned *index* used with the **in** operator is parenthesized. The **in** operator, which tests for the existence of a particular array element, shall not cause that element to exist. Any other reference to a nonexistent array element shall automatically create it.

Comparisons (with the ' $<$ ', ' $<=$ ', ' $!=$ ', ' $==$ ', ' $>$ ', and ' $>=$ ' operators) shall be made numerically if both operands are numeric, if one is numeric and the other has a string value that is a numeric string, or if one is numeric and the other has the uninitialized value. Otherwise, operands shall be converted to strings as required and a string comparison shall be made using the locale-specific collation sequence. The value of the comparison expression shall be 1 if the relation is true, or 0 if the relation is false.

Variables and Special Variables

Variables can be used in an *awk* program by referencing them. With the exception of function parameters (see [User-Defined Functions](#), on page 2446), they are not explicitly declared. Function parameter names shall be local to the function; all other variable names shall be global. The same name shall not be used as both a function parameter name and as the name of a function or a special *awk* variable. The same name shall not be used both as a variable name with global scope and as the name of a function. The same name shall not be used within the same scope both as a scalar variable and as an array. Uninitialized variables, including scalar variables, array elements, and field variables, shall have an uninitialized value. An uninitialized value shall have both a numeric value of zero and a string value of the empty string. Evaluation of variables with an uninitialized value, to either string or numeric, shall be determined by the context in which they are used.

Field variables shall be designated by a ' $\$$ ' followed by a number or numerical expression. The effect of the field number *expression* evaluating to anything other than a non-negative integer is unspecified; uninitialized variables or string values need not be converted to numeric values in this context. New field variables can be created by assigning a value to them. References to nonexistent fields (that is, fields after **\$NF**), shall evaluate to the uninitialized value. Such references shall not create new fields. However, assigning to a nonexistent field (for example, $\$(NF+2)=5$) shall increase the value of **NF**; create any intervening fields with the uninitialized value; and cause the value of **\$0** to be recomputed, with the fields being separated by the value of **OFS**. Each field variable shall have a string value or an uninitialized value when created. Field variables shall have the uninitialized value when created from **\$0** using **FS** and the variable does not contain any characters. If appropriate, the field variable shall be considered a numeric string (see [Expressions in awk](#), on page 2433).

Implementations shall support the following other special variables that are set by *awk*:

ARGC The number of elements in the **ARGV** array.

ARGV An array of command line arguments, excluding options and the *program* argument, numbered from zero to **ARGC**–1.

The arguments in **ARGV** can be modified or added to; **ARGC** can be altered. As each input file ends, *awk* shall treat the next non-null element of **ARGV**, up to the current value of **ARGC**–1, inclusive, as the name of the next input file. Thus, setting an element of **ARGV** to null means that it shall not be treated as an input file. The name ' $-$ ' indicates the standard input. If an argument matches the format of an *assignment* operand, this argument shall be treated as an *assignment* rather than a *file* argument.

awk

Utilities

77420	CONVFMT	The printf format for converting numbers to strings (except for output statements, where OFMT is used); "% . 6g" by default.
77421		
77422	ENVIRON	An array representing the value of the environment, as described in the <i>exec</i> functions defined in the System Interfaces volume of POSIX.1-2008. The indices of the array shall be strings consisting of the names of the environment variables, and the value of each array element shall be a string consisting of the value of that variable. If appropriate, the environment variable shall be considered a <i>numeric string</i> (see Expressions in awk , on page 2433); the array element shall also have its numeric value.
77423		
77424		
77425		
77426		
77427		
77428		
77429		In all cases where the behavior of <i>awk</i> is affected by environment variables (including the environment of any commands that <i>awk</i> executes via the system function or via pipeline redirections with the print statement, the printf statement, or the getline function), the environment used shall be the environment at the time <i>awk</i> began executing; it is implementation-defined whether any modification of ENVIRON affects this environment.
77430		
77431		
77432		
77433		
77434		
77435	FILENAME	A pathname of the current input file. Inside a BEGIN action the value is undefined. Inside an END action the value shall be the name of the last input file processed.
77436		
77437		
77438	FNR	The ordinal number of the current record in the current file. Inside a BEGIN action the value shall be zero. Inside an END action the value shall be the number of the last record processed in the last file processed.
77439		
77440		
77441	FS	Input field separator regular expression; a <space> by default.
77442	NF	The number of fields in the current record. Inside a BEGIN action, the use of NF is undefined unless a getline function without a <i>var</i> argument is executed previously. Inside an END action, NF shall retain the value it had for the last record read, unless a subsequent redirected, getline function without a <i>var</i> argument is performed prior to entering the END action.
77443		
77444		
77445		
77446		
77447	NR	The ordinal number of the current record from the start of input. Inside a BEGIN action the value shall be zero. Inside an END action the value shall be the number of the last record processed.
77448		
77449		
77450	OFMT	The printf format for converting numbers to strings in output statements (see Output Statements , on page 2442); "% . 6g" by default. The result of the conversion is unspecified if the value of OFMT is not a floating-point format specification.
77451		
77452		
77453	OFS	The print statement output field separator; <space> by default.
77454	ORS	The print statement output record separator; a <newline> by default.
77455	RLENGTH	The length of the string matched by the match function.
77456	RS	The first character of the string value of RS shall be the input record separator; a <newline> by default. If RS contains more than one character, the results are unspecified. If RS is null, then records are separated by sequences consisting of a <newline> plus one or more blank lines, leading or trailing blank lines shall not result in empty records at the beginning or end of the input, and a <newline> shall always be a field separator, no matter what the value of FS is.
77457		
77458		
77459		
77460		
77461		
77462	RSTART	The starting position of the string matched by the match function, numbering from 1. This shall always be equivalent to the return value of the match function.
77463		

SUBSEP The subscript separator string for multi-dimensional arrays; the default value is implementation-defined.

Regular Expressions

The *awk* utility shall make use of the extended regular expression notation (see XBD [Section 9.4](#), on page 188) except that it shall allow the use of C-language conventions for escaping special characters within the EREs, as specified in the table in XBD [Chapter 5](#) (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v') and the following table; these escape sequences shall be recognized both inside and outside bracket expressions. Note that records need not be separated by <newline> characters and string constants can contain <newline> characters, so even the "\n" sequence is valid in *awk* EREs. Using a <slash> character within an ERE requires the escaping shown in the following table.

Table 4-2 Escape Sequences in *awk*

Escape Sequence	Description	Meaning
\ "	<backslash> <quotation-mark>	<quotation-mark> character
\ /	<backslash> <slash>	<slash> character
\ddd	A <backslash> character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <backslash> for each byte.
\c	A <backslash> character followed by any character not described in this table or in the table in XBD Chapter 5 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').	Undefined

A regular expression can be matched against a specific field or string by using one of the two regular expression matching operators, '~' and '!~'. These operators shall interpret their right-hand operand as a regular expression and their left-hand operand as a string. If the regular expression matches the string, the '~' expression shall evaluate to a value of 1, and the '!~' expression shall evaluate to a value of 0. (The regular expression matching operation is as defined by the term matched in XBD [Section 9.1](#) (on page 181), where a match occurs on any part of the string unless the regular expression is limited with the <circumflex> or <dollar-sign> special characters.) If the regular expression does not match the string, the '~' expression shall evaluate to a value of 0, and the '!~' expression shall evaluate to a value of 1. If the right-hand operand is any expression other than the lexical token **ERE**, the string value of the expression shall be interpreted as an extended regular expression, including the escape conventions described above. Note that these same escape conventions shall also be applied in determining the value of a string literal (the lexical token **STRING**), and thus shall be applied a second time when a string literal is used in this context.

When an **ERE** token appears as an expression in any context other than as the right-hand of the '~' or '!~' operator or as one of the built-in function arguments described below, the value of the resulting expression shall be the equivalent of:

77509 \$0 ~ /ere/

77510 The *ere* argument to the **gsub**, **match**, **sub** functions, and the *fs* argument to the **split** function
 77511 (see [String Functions](#), on page 2444) shall be interpreted as extended regular expressions. These
 77512 can be either **ERE** tokens or arbitrary expressions, and shall be interpreted in the same manner
 77513 as the right-hand side of the ' ~ ' or " ! ~ " operator.

77514 An extended regular expression can be used to separate fields by using the **-F ERE** option or by
 77515 assigning a string containing the expression to the built-in variable **FS**. The default value of the
 77516 **FS** variable shall be a single <space>. The following describes **FS** behavior:

- 77517 1. If **FS** is a null string, the behavior is unspecified.
- 77518 2. If **FS** is a single character:
 - 77519 a. If **FS** is <space>, skip leading and trailing <blank> characters; fields shall be
 77520 delimited by sets of one or more <blank> characters.
 - 77521 b. Otherwise, if **FS** is any other character *c*, fields shall be delimited by each single
 77522 occurrence of *c*.
- 77523 3. Otherwise, the string value of **FS** shall be considered to be an extended regular
 77524 expression. Each occurrence of a sequence matching the extended regular expression shall
 77525 delimit fields.

77526 Except for the ' ~ ' and " ! ~ " operators, and in the **gsub**, **match**, **split**, and **sub** built-in functions,
 77527 ERE matching shall be based on input records; that is, record separator characters (the first
 77528 character of the value of the variable **RS**, <newline> by default) cannot be embedded in the
 77529 expression, and no expression shall match the record separator character. If the record separator
 77530 is not <newline>, <newline> characters embedded in the expression can be matched. For the
 77531 ' ~ ' and " ! ~ " operators, and in those four built-in functions, ERE matching shall be based on
 77532 text strings; that is, any character (including <newline> and the record separator) can be
 77533 embedded in the pattern, and an appropriate pattern shall match any character. However, in all
 77534 *awk* ERE matching, the use of one or more NUL characters in the pattern, input record, or text
 77535 string produces undefined results.

77536 Patterns

77537 A *pattern* is any valid *expression*, a range specified by two expressions separated by a comma, or
 77538 one of the two special patterns **BEGIN** or **END**.

77539 Special Patterns

77540 The *awk* utility shall recognize two special patterns, **BEGIN** and **END**. Each **BEGIN** pattern
 77541 shall be matched once and its associated action executed before the first record of input is read—
 77542 except possibly by use of the **getline** function (see [Input/Output and General Functions](#), on
 77543 page 2445) in a prior **BEGIN** action—and before command line assignment is done. Each **END**
 77544 pattern shall be matched once and its associated action executed after the last record of input has
 77545 been read. These two patterns shall have associated actions.

77546 **BEGIN** and **END** shall not combine with other patterns. Multiple **BEGIN** and **END** patterns
 77547 shall be allowed. The actions associated with the **BEGIN** patterns shall be executed in the order
 77548 specified in the program, as are the **END** actions. An **END** pattern can precede a **BEGIN** pattern
 77549 in a program.

77550 If an *awk* program consists of only actions with the pattern **BEGIN**, and the **BEGIN** action
 77551 contains no **getline** function, *awk* shall exit without reading its input when the last statement in
 77552 the last **BEGIN** action is executed. If an *awk* program consists of only actions with the pattern

77553 **END** or only actions with the patterns **BEGIN** and **END**, the input shall be read before the
77554 statements in the **END** actions are executed.

77555 Expression Patterns

77556 An expression pattern shall be evaluated as if it were an expression in a Boolean context. If the
77557 result is true, the pattern shall be considered to match, and the associated action (if any) shall be
77558 executed. If the result is false, the action shall not be executed.

77559 Pattern Ranges

77560 A pattern range consists of two expressions separated by a comma; in this case, the action shall
77561 be performed for all records between a match of the first expression and the following match of
77562 the second expression, inclusive. At this point, the pattern range can be repeated starting at
77563 input records subsequent to the end of the matched range.

77564 Actions

77565 An action is a sequence of statements as shown in the grammar in [Grammar](#) (on page 2447).
77566 Any single statement can be replaced by a statement list enclosed in curly braces. The
77567 application shall ensure that statements in a statement list are separated by <newline> or
77568 <semicolon> characters. Statements in a statement list shall be executed sequentially in the order
77569 that they appear.

77570 The *expression* acting as the conditional in an **if** statement shall be evaluated and if it is non-zero
77571 or non-null, the following statement shall be executed; otherwise, if **else** is present, the statement
77572 following the **else** shall be executed.

77573 The **if**, **while**, **do...while**, **for**, **break**, and **continue** statements are based on the ISO C standard
77574 (see [Section 1.1.2](#), on page 2283), except that the Boolean expressions shall be treated as
77575 described in [Expressions in awk](#) (on page 2433), and except in the case of:

77576 `for (variable in array)`

77577 which shall iterate, assigning each *index* of *array* to *variable* in an unspecified order. The results of
77578 adding new elements to *array* within such a **for** loop are undefined. If a **break** or **continue**
77579 statement occurs outside of a loop, the behavior is undefined.

77580 The **delete** statement shall remove an individual array element. Thus, the following code deletes
77581 an entire array:

77582 `for (index in array)`
77583 `delete array[index]`

77584 The **next** statement shall cause all further processing of the current input record to be
77585 abandoned. The behavior is undefined if a **next** statement appears or is invoked in a **BEGIN** or
77586 **END** action.

77587 The **exit** statement shall invoke all **END** actions in the order in which they occur in the program
77588 source and then terminate the program without reading further input. An **exit** statement inside
77589 an **END** action shall terminate the program without further execution of **END** actions. If an
77590 expression is specified in an **exit** statement, its numeric value shall be the exit status of *awk*,
77591 unless subsequent errors are encountered or a subsequent **exit** statement with an expression is
77592 executed.

Output Statements

Both **print** and **printf** statements shall write to standard output by default. The output shall be written to the location specified by *output_redirection* if one is supplied, as follows:

```
> expression
>> expression
| expression
```

In all cases, the *expression* shall be evaluated to produce a string that is used as a pathname into which to write (for '*>*' or "*>>*") or as a command to be executed (for '*|*'). Using the first two forms, if the file of that name is not currently open, it shall be opened, creating it if necessary and using the first form, truncating the file. The output then shall be appended to the file. As long as the file remains open, subsequent calls in which *expression* evaluates to the same string value shall simply append output to the file. The file remains open until the **close** function (see [Input/Output and General Functions](#), on page 2445) is called with an expression that evaluates to the same string value.

The third form shall write output onto a stream piped to the input of a command. The stream shall be created if no stream is currently open with the value of *expression* as its command name. The stream created shall be equivalent to one created by a call to the *popen()* function defined in the System Interfaces volume of POSIX.1-2008 with the value of *expression* as the *command* argument and a value of *w* as the *mode* argument. As long as the stream remains open, subsequent calls in which *expression* evaluates to the same string value shall write output to the existing stream. The stream shall remain open until the **close** function (see [Input/Output and General Functions](#), on page 2445) is called with an expression that evaluates to the same string value. At that time, the stream shall be closed as if by a call to the *pclose()* function defined in the System Interfaces volume of POSIX.1-2008.

As described in detail by the grammar in [Grammar](#) (on page 2447), these output statements shall take a <comma>-separated list of *expressions* referred to in the grammar by the non-terminal symbols **expr_list**, **print_expr_list**, or **print_expr_list_opt**. This list is referred to here as the *expression list*, and each member is referred to as an *expression argument*.

The **print** statement shall write the value of each expression argument onto the indicated output stream separated by the current output field separator (see variable **OFS** above), and terminated by the output record separator (see variable **ORS** above). All expression arguments shall be taken as strings, being converted if necessary; this conversion shall be as described in [Expressions in awk](#) (on page 2433), with the exception that the **printf** format in **OFMT** shall be used instead of the value in **CONVFMT**. An empty expression list shall stand for the whole input record (\$0).

The **printf** statement shall produce output based on a notation similar to the File Format Notation used to describe file formats in this volume of POSIX.1-2008 (see XBD [Chapter 5](#), on page 121). Output shall be produced as specified with the first *expression* argument as the string *format* and subsequent *expression* arguments as the strings *arg1* to *argn*, inclusive, with the following exceptions:

1. The *format* shall be an actual character string rather than a graphical representation. Therefore, it cannot contain empty character positions. The <space> in the *format* string, in any context other than a *flag* of a conversion specification, shall be treated as an ordinary character that is copied to the output.
2. If the character set contains a 'Δ' character and that character appears in the *format* string, it shall be treated as an ordinary character that is copied to the output.

3. The *escape sequences* beginning with a <backslash> character shall be treated as sequences of ordinary characters that are copied to the output. Note that these same sequences shall be interpreted lexically by *awk* when they appear in literal strings, but they shall not be treated specially by the **printf** statement.
4. A *field width* or *precision* can be specified as the ' * ' character instead of a digit string. In this case the next argument from the expression list shall be fetched and its numeric value taken as the field width or precision.
5. The implementation shall not precede or follow output from the *d* or *u* conversion specifier characters with <blank> characters not specified by the *format* string.
6. The implementation shall not precede output from the *o* conversion specifier character with leading zeros not specified by the *format* string.
7. For the *c* conversion specifier character: if the argument has a numeric value, the character whose encoding is that value shall be output. If the value is zero or is not the encoding of any character in the character set, the behavior is undefined. If the argument does not have a numeric value, the first character of the string value shall be output; if the string does not contain any characters, the behavior is undefined.
8. For each conversion specification that consumes an argument, the next expression argument shall be evaluated. With the exception of the *c* conversion specifier character, the value shall be converted (according to the rules specified in *Expressions in awk*, on page 2433) to the appropriate type for the conversion specification.
9. If there are insufficient expression arguments to satisfy all the conversion specifications in the *format* string, the behavior is undefined.
10. If any character sequence in the *format* string begins with a ' % ' character, but does not form a valid conversion specification, the behavior is unspecified.

Both **print** and **printf** can output at least {LINE_MAX} bytes.

Functions

The *awk* language has a variety of built-in functions: arithmetic, string, input/output, and general.

Arithmetic Functions

The arithmetic functions, except for **int**, shall be based on the ISO C standard (see *Section 1.1.2*, on page 2283). The behavior is undefined in cases where the ISO C standard specifies that an error be returned or that the behavior is undefined. Although the grammar (see *Grammar*, on page 2447) permits built-in functions to appear with no arguments or parentheses, unless the argument or parentheses are indicated as optional in the following list (by displaying them within the " [] " brackets), such use is undefined.

atan2 (<i>y</i> , <i>x</i>)	Return arctangent of <i>y</i> / <i>x</i> in radians in the range $[-\pi, \pi]$.
cos (<i>x</i>)	Return cosine of <i>x</i> , where <i>x</i> is in radians.
sin (<i>x</i>)	Return sine of <i>x</i> , where <i>x</i> is in radians.
exp (<i>x</i>)	Return the exponential function of <i>x</i> .
log (<i>x</i>)	Return the natural logarithm of <i>x</i> .

77679	sqrt (<i>x</i>)	Return the square root of <i>x</i> .
77680	int (<i>x</i>)	Return the argument truncated to an integer. Truncation shall be toward 0 when <i>x</i> >0.
77681		
77682	rand ()	Return a random number <i>n</i> , such that $0 \leq n < 1$.
77683	srand (<i>expr</i>)	Set the seed value for <i>rand</i> to <i>expr</i> or use the time of day if <i>expr</i> is omitted. The previous seed value shall be returned.
77684		

77685 String Functions

77686 The string functions in the following list shall be supported. Although the grammar (see
77687 [Grammar](#), on page 2447) permits built-in functions to appear with no arguments or parentheses,
77688 unless the argument or parentheses are indicated as optional in the following list (by displaying
77689 them within the " [] " brackets), such use is undefined.

77690	gsub (<i>ere</i> , <i>repl</i> [, <i>in</i>])	
77691		Behave like sub (see below), except that it shall replace all occurrences of the
77692		regular expression (like the <i>ed</i> utility global substitute) in \$0 or in the <i>in</i> argument,
77693		when specified.
77694	index (<i>s</i> , <i>t</i>)	Return the position, in characters, numbering from 1, in string <i>s</i> where string <i>t</i> first
77695		occurs, or zero if it does not occur at all.
77696	length [(<i>[s]</i>)]	Return the length, in characters, of its argument taken as a string, or of the whole
77697		record, \$0, if there is no argument.
77698	match (<i>s</i> , <i>ere</i>)	Return the position, in characters, numbering from 1, in string <i>s</i> where the
77699		extended regular expression <i>ere</i> occurs, or zero if it does not occur at all. RSTART
77700		shall be set to the starting position (which is the same as the returned value), zero
77701		if no match is found; RLENGTH shall be set to the length of the matched string, -1
77702		if no match is found.
77703	split (<i>s</i> , <i>a</i> [, <i>fs</i>])	
77704		Split the string <i>s</i> into array elements <i>a</i> [1], <i>a</i> [2], ..., <i>a</i> [<i>n</i>], and return <i>n</i> . All elements
77705		of the array shall be deleted before the split is performed. The separation shall be
77706		done with the ERE <i>fs</i> or with the field separator FS if <i>fs</i> is not given. Each array
77707		element shall have a string value when created and, if appropriate, the array
77708		element shall be considered a numeric string (see Expressions in awk , on page
77709		2433). The effect of a null string as the value of <i>fs</i> is unspecified.
77710	sprintf (<i>fmt</i> , <i>expr</i> , <i>expr</i> , ...)	
77711		Format the expressions according to the printf format given by <i>fmt</i> and return the
77712		resulting string.
77713	sub (<i>ere</i> , <i>repl</i> [, <i>in</i>])	
77714		Substitute the string <i>repl</i> in place of the first instance of the extended regular
77715		expression <i>ERE</i> in string <i>in</i> and return the number of substitutions. An
77716		<ampersand> (' & ') appearing in the string <i>repl</i> shall be replaced by the string
77717		from <i>in</i> that matches the ERE. An <ampersand> preceded with a <backslash> shall
77718		be interpreted as the literal <ampersand> character. An occurrence of two
77719		consecutive <backslash> characters shall be interpreted as just a single literal
77720		<backslash> character. Any other occurrence of a <backslash> (for example,
77721		preceding any other character) shall be treated as a literal <backslash> character.
77722		Note that if <i>repl</i> is a string literal (the lexical token STRING ; see Grammar , on page
77723		2447), the handling of the <ampersand> character occurs after any lexical

77724 processing, including any lexical <backslash>-escape sequence processing. If *in* is
 77725 specified and it is not an lvalue (see [Expressions in awk](#), on page 2433), the
 77726 behavior is undefined. If *in* is omitted, *awk* shall use the current record (\$0) in its
 77727 place.

77728 **substr**(*s*, *m*[, *n*])

77729 Return the at most *n*-character substring of *s* that begins at position *m*, numbering
 77730 from 1. If *n* is omitted, or if *n* specifies more characters than are left in the string,
 77731 the length of the substring shall be limited by the length of the string *s*.

77732 **tolower**(*s*) Return a string based on the string *s*. Each character in *s* that is an uppercase letter
 77733 specified to have a **tolower** mapping by the *LC_CTYPE* category of the current
 77734 locale shall be replaced in the returned string by the lowercase letter specified by
 77735 the mapping. Other characters in *s* shall be unchanged in the returned string.

77736 **toupper**(*s*) Return a string based on the string *s*. Each character in *s* that is a lowercase letter
 77737 specified to have a **toupper** mapping by the *LC_CTYPE* category of the current
 77738 locale is replaced in the returned string by the uppercase letter specified by the
 77739 mapping. Other characters in *s* are unchanged in the returned string.

77740 All of the preceding functions that take *ERE* as a parameter expect a pattern or a string valued
 77741 expression that is a regular expression as defined in [Regular Expressions](#) (on page 2439).

77742 Input/Output and General Functions

77743 The input/output and general functions are:

77744 **close**(*expression*)

77745 Close the file or pipe opened by a **print** or **printf** statement or a call to **getline** with
 77746 the same string-valued *expression*. The limit on the number of open *expression*
 77747 arguments is implementation-defined. If the close was successful, the function
 77748 shall return zero; otherwise, it shall return non-zero.

77749 *expression* | **getline** [*var*]

77750 Read a record of input from a stream piped from the output of a command. The
 77751 stream shall be created if no stream is currently open with the value of *expression* as
 77752 its command name. The stream created shall be equivalent to one created by a call
 77753 to the *popen*() function with the value of *expression* as the *command* argument and a
 77754 value of *r* as the *mode* argument. As long as the stream remains open, subsequent
 77755 calls in which *expression* evaluates to the same string value shall read subsequent
 77756 records from the stream. The stream shall remain open until the **close** function is
 77757 called with an expression that evaluates to the same string value. At that time, the
 77758 stream shall be closed as if by a call to the *pclose*() function. If *var* is omitted, \$0 and
 77759 NF shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered
 77760 a numeric string (see [Expressions in awk](#), on page 2433).

77761 The **getline** operator can form ambiguous constructs when there are
 77762 unparenthesized operators (including concatenate) to the left of the ' | ' (to the
 77763 beginning of the expression containing **getline**). In the context of the ' \$ ' operator,
 77764 ' | ' shall behave as if it had a lower precedence than ' \$ '. The result of evaluating
 77765 other operators is unspecified, and conforming applications shall parenthesize
 77766 properly all such usages.

77767 **getline** Set \$0 to the next input record from the current input file. This form of **getline** shall
 77768 set the NF, NR, and FNR variables.

getline *var* Set variable *var* to the next input record from the current input file and, if appropriate, *var* shall be considered a numeric string (see [Expressions in awk](#), on page 2433). This form of **getline** shall set the **FNR** and **NR** variables.

getline [*var*] < *expression*

Read the next record of input from a named file. The *expression* shall be evaluated to produce a string that is used as a pathname. If the file of that name is not currently open, it shall be opened. As long as the stream remains open, subsequent calls in which *expression* evaluates to the same string value shall read subsequent records from the file. The file shall remain open until the **close** function is called with an expression that evaluates to the same string value. If *var* is omitted, **\$0** and **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered a numeric string (see [Expressions in awk](#), on page 2433).

The **getline** operator can form ambiguous constructs when there are unparenthesized binary operators (including concatenate) to the right of the '**<**' (up to the end of the expression containing the **getline**). The result of evaluating such a construct is unspecified, and conforming applications shall parenthesize properly all such usages.

system(*expression*)

Execute the command given by *expression* in a manner equivalent to the *system()* function defined in the System Interfaces volume of POSIX.1-2008 and return the exit status of the command.

All forms of **getline** shall return 1 for successful input, zero for end-of-file, and -1 for an error.

Where strings are used as the name of a file or pipeline, the application shall ensure that the strings are textually identical. The terminology "same string value" implies that "equivalent strings", even those that differ only by <space> characters, represent different files.

User-Defined Functions

The *awk* language also provides user-defined functions. Such functions can be defined as:

```
function name([parameter, ...]) { statements }
```

A function can be referred to anywhere in an *awk* program; in particular, its use can precede its definition. The scope of a function is global.

Function parameters, if present, can be either scalars or arrays; the behavior is undefined if an array name is passed as a parameter that the function uses as a scalar, or if a scalar expression is passed as a parameter that the function uses as an array. Function parameters shall be passed by value if scalar and by reference if array name.

The number of parameters in the function definition need not match the number of parameters in the function call. Excess formal parameters can be used as local variables. If fewer arguments are supplied in a function call than are in the function definition, the extra parameters that are used in the function body as scalars shall evaluate to the uninitialized value until they are otherwise initialized, and the extra parameters that are used in the function body as arrays shall be treated as uninitialized arrays where each element evaluates to the uninitialized value until otherwise initialized.

When invoking a function, no white space can be placed between the function name and the opening parenthesis. Function calls can be nested and recursive calls can be made upon functions. Upon return from any nested or recursive function call, the values of all of the calling function's parameters shall be unchanged, except for array parameters passed by reference. The

return statement can be used to return a value. If a **return** statement appears outside of a function definition, the behavior is undefined.

In the function definition, <newline> characters shall be optional before the opening brace and after the closing brace. Function definitions can appear anywhere in the program where a *pattern-action* pair is allowed.

Grammar

The grammar in this section and the lexical conventions in the following section shall together describe the syntax for *awk* programs. The general conventions for this style of grammar are described in [Section 1.3](#) (on page 2287). A valid program can be represented as the non-terminal symbol *program* in the grammar. This formal syntax shall take precedence over the preceding text syntax description.

```
%token NAME NUMBER STRING ERE
%token FUNC_NAME /* Name followed by '(' without white space. */

/* Keywords */
%token Begin End
/* 'BEGIN' 'END' */

%token Break Continue Delete Do Else
/* 'break' 'continue' 'delete' 'do' 'else' */

%token Exit For Function If In
/* 'exit' 'for' 'function' 'if' 'in' */

%token Next Print Printf Return While
/* 'next' 'print' 'printf' 'return' 'while' */

/* Reserved function names */
%token BUILTIN_FUNC_NAME
/* One token for the following:
 * atan2 cos sin exp log sqrt int rand srand
 * gsub index length match split sprintf sub
 * substr tolower toupper close system
 */

%token GETLINE
/* Syntactically different from other built-ins. */

/* Two-character tokens. */
%token ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN POW_ASSIGN
/* '+=' '-=' '*=' '/=' '%=' '^=' */

%token OR AND NO_MATCH EQ LE GE NE INCR DECR APPEND
/* '||' '&&' '!~' '==' '<=' '>=' '!=' '++' '--' '>>' */

/* One-character tokens. */
%token '{' '}' '(' ')' '[' ']' ',' ';' NEWLINE
%token '+' '-' '*' '%' '^' '!' '>' '<' '|' '?' ':' '~' '$' '='

%start program
%%

program      : item_list
              | actionless_item_list
              ;
```

```

77858     item_list      : newline_opt
77859                     | actionless_item_list item terminator
77860                     | item_list           item terminator
77861                     | item_list           action terminator
77862                     ;

77863     actionless_item_list : item_list           pattern terminator
77864                         | actionless_item_list pattern terminator
77865                         ;

77866     item              : pattern action
77867                       | Function NAME      '(' param_list_opt ')'
77868                           newline_opt action
77869                       | Function FUNC_NAME '(' param_list_opt ')'
77870                           newline_opt action
77871                       ;

77872     param_list_opt    : /* empty */
77873                       | param_list
77874                       ;

77875     param_list        : NAME
77876                       | param_list ',' NAME
77877                       ;

77878     pattern            : Begin
77879                       | End
77880                       | expr
77881                       | expr ',' newline_opt expr
77882                       ;

77883     action              : '{' newline_opt                '}'
77884                       | '{' newline_opt terminated_statement_list '}'
77885                       | '{' newline_opt unterminated_statement_list '}'
77886                       ;

77887     terminator         : terminator ';'
77888                       | terminator NEWLINE
77889                       | ';'
77890                       | NEWLINE
77891                       ;

77892     terminated_statement_list : terminated_statement
77893                             | terminated_statement_list terminated_statement
77894                             ;

77895     unterminated_statement_list : unterminated_statement
77896                             | terminated_statement_list unterminated_statement
77897                             ;

77898     terminated_statement : action newline_opt
77899                         | If '(' expr ')' newline_opt terminated_statement
77900                         | If '(' expr ')' newline_opt terminated_statement
77901                           Else newline_opt terminated_statement
77902                         | While '(' expr ')' newline_opt terminated_statement
77903                         | For '(' simple_statement_opt ';'
77904                           expr_opt ';' simple_statement_opt ')' newline_opt

```

Utilities

awk

```

77905         terminated_statement
77906     | For '(' NAME In NAME ')' newline_opt
77907         terminated_statement
77908     | ';' newline_opt
77909     | terminatable_statement NEWLINE newline_opt
77910     | terminatable_statement ';' newline_opt
77911     ;

77912     unterminated_statement : terminatable_statement
77913     | If '(' expr ')' newline_opt unterminated_statement
77914     | If '(' expr ')' newline_opt terminated_statement
77915     | Else newline_opt unterminated_statement
77916     | While '(' expr ')' newline_opt unterminated_statement
77917     | For '(' simple_statement_opt ';'
77918     |   expr_opt ';' simple_statement_opt ')' newline_opt
77919     |   unterminated_statement
77920     | For '(' NAME In NAME ')' newline_opt
77921     |   unterminated_statement
77922     ;

77923     terminatable_statement : simple_statement
77924     | Break
77925     | Continue
77926     | Next
77927     | Exit expr_opt
77928     | Return expr_opt
77929     | Do newline_opt terminated_statement While '(' expr ')'
77930     ;

77931     simple_statement_opt : /* empty */
77932     | simple_statement
77933     ;

77934     simple_statement : Delete NAME '[' expr_list ']'
77935     | expr
77936     | print_statement
77937     ;

77938     print_statement : simple_print_statement
77939     | simple_print_statement output_redirection
77940     ;

77941     simple_print_statement : Print print_expr_list_opt
77942     | Print '(' multiple_expr_list ')'
77943     | Printf print_expr_list
77944     | Printf '(' multiple_expr_list ')'
77945     ;

77946     output_redirection : '>' expr
77947     | APPEND expr
77948     | '|' expr
77949     ;

77950     expr_list_opt : /* empty */
77951     | expr_list
77952     ;

```

awk*Utilities*

```

77953      expr_list      : expr
77954                      | multiple_expr_list
77955                      ;

77956      multiple_expr_list : expr ',' newline_opt expr
77957                      | multiple_expr_list ',' newline_opt expr
77958                      ;

77959      expr_opt         : /* empty */
77960                      | expr
77961                      ;

77962      expr              : unary_expr
77963                      | non_unary_expr
77964                      ;

77965      unary_expr        : '+' expr
77966                      | '-' expr
77967                      | unary_expr '^'      expr
77968                      | unary_expr '*'      expr
77969                      | unary_expr '/'      expr
77970                      | unary_expr '%'      expr
77971                      | unary_expr '+'      expr
77972                      | unary_expr '-'      expr
77973                      | unary_expr          non_unary_expr
77974                      | unary_expr '<'      expr
77975                      | unary_expr LE      expr
77976                      | unary_expr NE      expr
77977                      | unary_expr EQ      expr
77978                      | unary_expr '>'      expr
77979                      | unary_expr GE      expr
77980                      | unary_expr '~'      expr
77981                      | unary_expr NO_MATCH expr
77982                      | unary_expr In NAME
77983                      | unary_expr AND newline_opt expr
77984                      | unary_expr OR  newline_opt expr
77985                      | unary_expr '?' expr ':' expr
77986                      | unary_input_function
77987                      ;

77988      non_unary_expr    : '(' expr ')'
77989                      | '!' expr
77990                      | non_unary_expr '^'      expr
77991                      | non_unary_expr '*'      expr
77992                      | non_unary_expr '/'      expr
77993                      | non_unary_expr '%'      expr
77994                      | non_unary_expr '+'      expr
77995                      | non_unary_expr '-'      expr
77996                      | non_unary_expr          non_unary_expr
77997                      | non_unary_expr '<'      expr
77998                      | non_unary_expr LE      expr
77999                      | non_unary_expr NE      expr
78000                      | non_unary_expr EQ      expr
78001                      | non_unary_expr '>'      expr

```

Utilities

awk

```

78002 | non_unary_expr GE      expr
78003 | non_unary_expr '~'     expr
78004 | non_unary_expr NO_MATCH expr
78005 | non_unary_expr In NAME
78006 | '(' multiple_expr_list ')' In NAME
78007 | non_unary_expr AND newline_opt expr
78008 | non_unary_expr OR  newline_opt expr
78009 | non_unary_expr '?' expr ':' expr
78010 | NUMBER
78011 | STRING
78012 | lvalue
78013 | ERE
78014 | lvalue INCR
78015 | lvalue DECR
78016 | INCR lvalue
78017 | DECR lvalue
78018 | lvalue POW_ASSIGN expr
78019 | lvalue MOD_ASSIGN expr
78020 | lvalue MUL_ASSIGN expr
78021 | lvalue DIV_ASSIGN expr
78022 | lvalue ADD_ASSIGN expr
78023 | lvalue SUB_ASSIGN expr
78024 | lvalue '=' expr
78025 | FUNC_NAME '(' expr_list_opt ')'
78026 | /* no white space allowed before '(' */
78027 | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
78028 | BUILTIN_FUNC_NAME
78029 | non_unary_input_function
78030 |
78031 print_expr_list_opt : /* empty */
78032 | print_expr_list
78033 |
78034 print_expr_list : print_expr
78035 | print_expr_list ',' newline_opt print_expr
78036 |
78037 print_expr : unary_print_expr
78038 | non_unary_print_expr
78039 |
78040 unary_print_expr : '+' print_expr
78041 | '-' print_expr
78042 | unary_print_expr '^' print_expr
78043 | unary_print_expr '*' print_expr
78044 | unary_print_expr '/' print_expr
78045 | unary_print_expr '%' print_expr
78046 | unary_print_expr '+' print_expr
78047 | unary_print_expr '-' print_expr
78048 | unary_print_expr non_unary_print_expr
78049 | unary_print_expr '~' print_expr
78050 | unary_print_expr NO_MATCH print_expr
78051 | unary_print_expr In NAME

```

awk*Utilities*

```

78052         | unary_print_expr AND newline_opt print_expr
78053         | unary_print_expr OR  newline_opt print_expr
78054         | unary_print_expr '?' print_expr ':' print_expr
78055         ;
78056 non_unary_print_expr : '(' expr ')'
78057         | '!' print_expr
78058         | non_unary_print_expr '^'      print_expr
78059         | non_unary_print_expr '*'      print_expr
78060         | non_unary_print_expr '/'      print_expr
78061         | non_unary_print_expr '%'      print_expr
78062         | non_unary_print_expr '+'      print_expr
78063         | non_unary_print_expr '-'      print_expr
78064         | non_unary_print_expr         non_unary_print_expr
78065         | non_unary_print_expr '~'      print_expr
78066         | non_unary_print_expr NO_MATCH print_expr
78067         | non_unary_print_expr In NAME
78068         | '(' multiple_expr_list ')' In NAME
78069         | non_unary_print_expr AND newline_opt print_expr
78070         | non_unary_print_expr OR  newline_opt print_expr
78071         | non_unary_print_expr '?' print_expr ':' print_expr
78072         | NUMBER
78073         | STRING
78074         | lvalue
78075         | ERE
78076         | lvalue INCR
78077         | lvalue DECR
78078         | INCR lvalue
78079         | DECR lvalue
78080         | lvalue POW_ASSIGN print_expr
78081         | lvalue MOD_ASSIGN print_expr
78082         | lvalue MUL_ASSIGN print_expr
78083         | lvalue DIV_ASSIGN print_expr
78084         | lvalue ADD_ASSIGN print_expr
78085         | lvalue SUB_ASSIGN print_expr
78086         | lvalue '=' print_expr
78087         | FUNC_NAME '(' expr_list_opt ')'
78088             /* no white space allowed before '(' */
78089         | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
78090         | BUILTIN_FUNC_NAME
78091         ;
78092 lvalue
78093         : NAME
78094         | NAME '[' expr_list ']'
78095         | '$' expr
78096         ;
78097 non_unary_input_function : simple_get
78098         | simple_get '<' expr
78099         | non_unary_expr '|' simple_get
78100         ;
78101 unary_input_function : unary_expr '|' simple_get
78102         ;

```



```

78102     simple_get      : GETLINE
78103                     | GETLINE lvalue
78104                     ;
78105     newline_opt      : /* empty */
78106                     | newline_opt NEWLINE
78107                     ;

```

This grammar has several ambiguities that shall be resolved as follows:

- Operator precedence and associativity shall be as described in Table 4-1 (on page 2433).
- In case of ambiguity, an **else** shall be associated with the most immediately preceding **if** that would satisfy the grammar.
- In some contexts, a <slash> ('/') that is used to surround an ERE could also be the division operator. This shall be resolved in such a way that wherever the division operator could appear, a <slash> is assumed to be the division operator. (There is no unary division operator.)

Each expression in an *awk* program shall conform to the precedence and associativity rules, even when this is not needed to resolve an ambiguity. For example, because '\$' has higher precedence than '++', the string "\$x++--" is not a valid *awk* expression, even though it is unambiguously parsed by the grammar as "\$ (x++) --".

One convention that might not be obvious from the formal grammar is where <newline> characters are acceptable. There are several obvious placements such as terminating a statement, and a <backslash> can be used to escape <newline> characters between any lexical tokens. In addition, <newline> characters without <backslash> characters can follow a comma, an open brace, logical AND operator ("&&"), logical OR operator ("||"), the **do** keyword, the **else** keyword, and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```

78126 { print $1,
78127       $2 }

```

Lexical Conventions

The lexical conventions for *awk* programs, with respect to the preceding grammar, shall be as follows:

1. Except as noted, *awk* shall recognize the longest possible token or delimiter beginning at a given point.
2. A comment shall consist of any characters beginning with the <number-sign> character and terminated by, but excluding the next occurrence of, a <newline>. Comments shall have no effect, except to delimit lexical tokens.
3. The <newline> shall be recognized as the token **NEWLINE**.
4. A <backslash> character immediately followed by a <newline> shall have no effect.
5. The token **STRING** shall represent a string constant. A string constant shall begin with the character ' '. Within a string constant, a <backslash> character shall be considered to begin an escape sequence as specified in the table in XBD Chapter 5 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'). In addition, the escape sequences in Table 4-2 (on page 2439) shall be recognized. A <newline> shall not occur within a string constant. A string constant shall be terminated by the first unescaped occurrence of the character ' ' after the one that begins the string constant. The value of the string

- 78145 shall be the sequence of all unescaped characters and values of escape sequences
78146 between, but not including, the two delimiting ' ' characters.
- 78147 6. The token **ERE** represents an extended regular expression constant. An ERE constant
78148 shall begin with the <slash> character. Within an ERE constant, a <backslash> character
78149 shall be considered to begin an escape sequence as specified in the table in XBD [Chapter 5](#)
78150 (on page 121). In addition, the escape sequences in [Table 4-2](#) (on page 2439) shall be
78151 recognized. The application shall ensure that a <newline> does not occur within an ERE
78152 constant. An ERE constant shall be terminated by the first unescaped occurrence of the
78153 <slash> character after the one that begins the ERE constant. The extended regular
78154 expression represented by the ERE constant shall be the sequence of all unescaped
78155 characters and values of escape sequences between, but not including, the two delimiting
78156 <slash> characters.
- 78157 7. A <blank> shall have no effect, except to delimit lexical tokens or within **STRING** or **ERE**
78158 tokens.
- 78159 8. The token **NUMBER** shall represent a numeric constant. Its form and numeric value shall
78160 either be equivalent to the **decimal-floating-constant** token as specified by the ISO C
78161 standard, or it shall be a sequence of decimal digits and shall be evaluated as an integer
78162 constant in decimal. In addition, implementations may accept numeric constants with the
78163 form and numeric value equivalent to the **hexadecimal-constant** and **hexadecimal-**
78164 **floating-constant** tokens as specified by the ISO C standard.
- 78165 If the value is too large or too small to be representable (see [Section 1.1.2](#), on page 2283),
78166 the behavior is undefined.
- 78167 9. A sequence of underscores, digits, and alphabetic characters from the portable character set (see
78168 XBD [Section 6.1](#), on page 125), beginning with an <underscore> or alphabetic character,
78169 shall be considered a word.
- 78170 10. The following words are keywords that shall be recognized as individual tokens; the
78171 name of the token is the same as the keyword:
- | | | | | | |
|-----------------|---------------|-------------|-----------------|--------------|---------------|
| BEGIN | delete | END | function | in | printf |
| break | do | exit | getline | next | return |
| continue | else | for | if | print | while |
- 78175 11. The following words are names of built-in functions and shall be recognized as the token
78176 **BUILTIN_FUNC_NAME**:
- | | | | | | |
|--------------|---------------|--------------|----------------|----------------|----------------|
| atan2 | gsub | log | split | sub | toupper |
| close | index | match | sprintf | substr | |
| cos | int | rand | sqr | system | |
| exp | length | sin | srand | tolower | |
- 78181 The above-listed keywords and names of built-in functions are considered reserved
78182 words.
- 78183 12. The token **NAME** shall consist of a word that is not a keyword or a name of a built-in
78184 function and is not followed immediately (without any delimiters) by the ' (' character.
- 78185 13. The token **FUNC_NAME** shall consist of a word that is not a keyword or a name of a
78186 built-in function, followed immediately (without any delimiters) by the ' (' character.
78187 The ' (' character shall not be included as part of the token.

14. The following two-character sequences shall be recognized as the named tokens:

Token Name	Sequence	Token Name	Sequence
ADD_ASSIGN	+=	NO_MATCH	!~
SUB_ASSIGN	--	EQ	==
MUL_ASSIGN	*=	LE	<=
DIV_ASSIGN	/=	GE	>=
MOD_ASSIGN	%=	NE	!=
POW_ASSIGN	^=	INCR	++
OR		DECR	--
AND	&&	APPEND	>>

15. The following single characters shall be recognized as tokens whose names are the character:

<newline> { } () [] , ; + - * % ^ ! > < | ? : ~ \$ =

There is a lexical ambiguity between the token **ERE** and the tokens **'/'** and **DIV_ASSIGN**. When an input sequence begins with a <slash> character in any syntactic context where the token **'/'** or **DIV_ASSIGN** could appear as the next token in a valid program, the longer of those two tokens that can be recognized shall be recognized. In any other syntactic context where the token **ERE** could appear as the next token in a valid program, the token **ERE** shall be recognized.

EXIT STATUS

The following exit values shall be returned:

- 0 All input files were processed successfully.
- >0 An error occurred.

The exit status can be altered within the program by using an **exit** expression.

CONSEQUENCES OF ERRORS

If any *file* operand is specified and the named file cannot be accessed, *awk* shall write a diagnostic message to standard error and terminate without any further action.

If the program specified by either the *program* operand or a *progfile* operand is not a valid *awk* program (as specified in the EXTENDED DESCRIPTION section), the behavior is undefined.

APPLICATION USAGE

The **index**, **length**, **match**, and **substr** functions should not be confused with similar functions in the ISO C standard; the *awk* versions deal with characters, while the ISO C standard deals with bytes.

Because the concatenation operation is represented by adjacent expressions rather than an explicit operator, it is often necessary to use parentheses to enforce the proper evaluation precedence.

EXAMPLES

The *awk* program specified in the command line is most easily specified within single-quotes (for example, *'program'*) for applications using *sh*, because *awk* programs commonly contain characters that are special to the shell, including double-quotes. In the cases where an *awk* program contains single-quote characters, it is usually easiest to specify most of the program as strings within single-quotes concatenated by the shell with quoted single-quote characters. For example:

```
awk '/'\''/{ print "quote:", $0 }'
```

prints all lines from the standard input containing a single-quote character, prefixed with *quote*..

The following are examples of simple *awk* programs:

1. Write to the standard output all input lines for which field 3 is greater than 5:

```
$3 > 5
```

2. Write every tenth line:

```
(NR % 10) == 0
```

3. Write any line with a substring matching the regular expression:

```
/(G|D)(2[0-9][[:alpha:]]*)/
```

4. Print any line with a substring containing a 'G' or 'D', followed by a sequence of digits and characters. This example uses character classes **digit** and **alpha** to match language-independent digit and alphabetic characters respectively:

```
/(G|D)([[:digit:]][[:alpha:]]*)/
```

5. Write any line in which the second field matches the regular expression and the fourth field does not:

```
$2 ~ /xyz/ && $4 !~ /xyz/
```

6. Write any line in which the second field contains a <backslash>:

```
$2 ~ /\\"/>

```

14. Write fields in reverse order, one per line (many lines out for each line in):

```
{ for (i = NF; i > 0; --i) print $i }
```

15. Write all lines between occurrences of the strings **start** and **stop**:

```
/start/, /stop/
```

16. Write all lines whose first field is different from the previous one:

```
$1 != prev { print; prev = $1 }
```

17. Simulate *echo*:

```
BEGIN {
    for (i = 1; i < ARGV; ++i)
        printf("%s%s", ARGV[i], i==ARGC-1?"\n":" ")
}
```

18. Write the path prefixes contained in the *PATH* environment variable, one per line:

```
BEGIN {
    n = split (ENVIRON["PATH"], path, ":")
    for (i = 1; i <= n; ++i)
        print path[i]
}
```

19. If there is a file named **input** containing page headers of the form:

Page #

and a file named **program** that contains:

```
/Page/    { $2 = n++; }
           { print }
```

then the command line:

```
awk -f program n=5 input
```

prints the file **input**, filling in page numbers starting at 5.

RATIONALE

This description is based on the new *awk*, “nawk”, (see the referenced *The AWK Programming Language*), which introduced a number of new features to the historical *awk*:

1. New keywords: **delete**, **do**, **function**, **return**
2. New built-in functions: **atan2**, **close**, **cos**, **gsub**, **match**, **rand**, **sin**, **srand**, **sub**, **system**
3. New predefined variables: **FNR**, **ARGC**, **ARGV**, **RSTART**, **RLENGTH**, **SUBSEP**
4. New expression operators: **?**, **:**, **..**, **^**
5. The **FS** variable and the third argument to **split**, now treated as extended regular expressions.
6. The operator precedence, changed to more closely match the C language. Two examples of code that operate differently are:

```
while ( n /= 10 > 1) ...
if (!"wk" ~ /bwk/) ...
```

Several features have been added based on newer implementations of *awk*:

- Multiple instances of `-f progfile` are permitted.
- The new option `-v assignment`.
- The new predefined variable **ENVIRON**.
- New built-in functions **toupper** and **tolower**.
- More formatting capabilities are added to **printf** to match the ISO C standard.

The overall *awk* syntax has always been based on the C language, with a few features from the shell command language and other sources. Because of this, it is not completely compatible with any other language, which has caused confusion for some users. It is not the intent of the standard developers to address such issues. A few relatively minor changes toward making the language more compatible with the ISO C standard were made; most of these changes are based on similar changes in recent implementations, as described above. There remain several C-language conventions that are not in *awk*. One of the notable ones is the `<comma>` operator, which is commonly used to specify multiple expressions in the C language **for** statement. Also, there are various places where *awk* is more restrictive than the C language regarding the type of expression that can be used in a given context. These limitations are due to the different features that the *awk* language does provide.

Regular expressions in *awk* have been extended somewhat from historical implementations to make them a pure superset of extended regular expressions, as defined by POSIX.1-2008 (see XBD Section 9.4, on page 188). The main extensions are internationalization features and interval expressions. Historical implementations of *awk* have long supported `<backslash>`-escape sequences as an extension to extended regular expressions, and this extension has been retained despite inconsistency with other utilities. The number of escape sequences recognized in both extended regular expressions and strings has varied (generally increasing with time) among implementations. The set specified by POSIX.1-2008 includes most sequences known to be supported by popular implementations and by the ISO C standard. One sequence that is not supported is hexadecimal value escapes beginning with `'\x'`. This would allow values expressed in more than 9 bits to be used within *awk* as in the ISO C standard. However, because this syntax has a non-deterministic length, it does not permit the subsequent character to be a hexadecimal digit. This limitation can be dealt with in the C language by the use of lexical string concatenation. In the *awk* language, concatenation could also be a solution for strings, but not for extended regular expressions (either lexical ERE tokens or strings used dynamically as regular expressions). Because of this limitation, the feature has not been added to POSIX.1-2008.

When a string variable is used in a context where an extended regular expression normally appears (where the lexical token ERE is used in the grammar) the string does not contain the literal `<slash>` characters.

Some versions of *awk* allow the form:

```
func name(args, ... ) { statements }
```

This has been deprecated by the authors of the language, who asked that it not be specified.

Historical implementations of *awk* produce an error if a **next** statement is executed in a **BEGIN** action, and cause *awk* to terminate if a **next** statement is executed in an **END** action. This behavior has not been documented, and it was not believed that it was necessary to standardize it.

The specification of conversions between string and numeric values is much more detailed than in the documentation of historical implementations or in the referenced *The AWK Programming Language*. Although most of the behavior is designed to be intuitive, the details are necessary to

ensure compatible behavior from different implementations. This is especially important in relational expressions since the types of the operands determine whether a string or numeric comparison is performed. From the perspective of an application developer, it is usually sufficient to expect intuitive behavior and to force conversions (by adding zero or concatenating a null string) when the type of an expression does not obviously match what is needed. The intent has been to specify historical practice in almost all cases. The one exception is that, in historical implementations, variables and constants maintain both string and numeric values after their original value is converted by any use. This means that referencing a variable or constant can have unexpected side-effects. For example, with historical implementations the following program:

```
{
    a = "+2"
    b = 2
    if (NR % 2)
        c = a + b
    if (a == b)
        print "numeric comparison"
    else
        print "string comparison"
}
```

would perform a numeric comparison (and output numeric comparison) for each odd-numbered line, but perform a string comparison (and output string comparison) for each even-numbered line. POSIX.1-2008 ensures that comparisons will be numeric if necessary. With historical implementations, the following program:

```
BEGIN {
    OFMT = "%e"
    print 3.14
    OFMT = "%f"
    print 3.14
}
```

would output "3.140000e+00" twice, because in the second **print** statement the constant "3.14" would have a string value from the previous conversion. POSIX.1-2008 requires that the output of the second **print** statement be "3.140000". The behavior of historical implementations was seen as too unintuitive and unpredictable.

It was pointed out that with the rules contained in early drafts, the following script would print nothing:

```
BEGIN {
    y[1.5] = 1
    OFMT = "%e"
    print y[1.5]
}
```

Therefore, a new variable, **CONVFMT**, was introduced. The **OFMT** variable is now restricted to affecting output conversions of numbers to strings and **CONVFMT** is used for internal conversions, such as comparisons or array indexing. The default value is the same as that for **OFMT**, so unless a program changes **CONVFMT** (which no historical program would do), it will receive the historical behavior associated with internal string conversions.

The POSIX *awk* lexical and syntactic conventions are specified more formally than in other sources. Again the intent has been to specify historical practice. One convention that may not be

obvious from the formal grammar as in other verbal descriptions is where <newline> characters are acceptable. There are several obvious placements such as terminating a statement, and a <backslash> can be used to escape <newline> characters between any lexical tokens. In addition, <newline> characters without <backslash> characters can follow a comma, an open brace, a logical AND operator ("&&"), a logical OR operator ("||"), the **do** keyword, the **else** keyword, and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```
{ print $1,
    $2 }
```

The requirement that *awk* add a trailing <newline> to the program argument text is to simplify the grammar, making it match a text file in form. There is no way for an application or test suite to determine whether a literal <newline> is added or whether *awk* simply acts as if it did.

POSIX.1-2008 requires several changes from historical implementations in order to support internationalization. Probably the most subtle of these is the use of the decimal-point character, defined by the *LC_NUMERIC* category of the locale, in representations of floating-point numbers. This locale-specific character is used in recognizing numeric input, in converting between strings and numeric values, and in formatting output. However, regardless of locale, the <period> character (the decimal-point character of the POSIX locale) is the decimal-point character recognized in processing *awk* programs (including assignments in command line arguments). This is essentially the same convention as the one used in the ISO C standard. The difference is that the C language includes the *setlocale()* function, which permits an application to modify its locale. Because of this capability, a C application begins executing with its locale set to the C locale, and only executes in the environment-specified locale after an explicit call to *setlocale()*. However, adding such an elaborate new feature to the *awk* language was seen as inappropriate for POSIX.1-2008. It is possible to execute an *awk* program explicitly in any desired locale by setting the environment in the shell.

The undefined behavior resulting from NULs in extended regular expressions allows future extensions for the GNU *gawk* program to process binary data.

The behavior in the case of invalid *awk* programs (including lexical, syntactic, and semantic errors) is undefined because it was considered overly limiting on implementations to specify. In most cases such errors can be expected to produce a diagnostic and a non-zero exit status. However, some implementations may choose to extend the language in ways that make use of certain invalid constructs. Other invalid constructs might be deemed worthy of a warning, but otherwise cause some reasonable behavior. Still other constructs may be very difficult to detect in some implementations. Also, different implementations might detect a given error during an initial parsing of the program (before reading any input files) while others might detect it when executing the program after reading some input. Implementors should be aware that diagnosing errors as early as possible and producing useful diagnostics can ease debugging of applications, and thus make an implementation more usable.

The unspecified behavior from using multi-character **RS** values is to allow possible future extensions based on extended regular expressions used for record separators. Historical implementations take the first character of the string and ignore the others.

Unspecified behavior when *split(string,array,<null>)* is used is to allow a proposed future extension that would split up a string into an array of individual characters.

In the context of the **getline** function, equally good arguments for different precedences of the **|** and **<** operators can be made. Historical practice has been that:

```
getline < "a" "b"
```

is parsed as:


```
( getline < "a" ) "b"
```

although many would argue that the intent was that the file **ab** should be read. However:

```
getline < "x" + 1
```

parses as:

```
getline < ( "x" + 1 )
```

Similar problems occur with the **|** version of **getline**, particularly in combination with **\$**. For example:

```
$"echo hi" | getline
```

(This situation is particularly problematic when used in a **print** statement, where the **|getline** part might be a redirection of the **print**.)

Since in most cases such constructs are not (or at least should not) be used (because they have a natural ambiguity for which there is no conventional parsing), the meaning of these constructs has been made explicitly unspecified. (The effect is that a conforming application that runs into the problem must parenthesize to resolve the ambiguity.) There appeared to be few if any actual uses of such constructs.

Grammars can be written that would cause an error under these circumstances. Where backwards-compatibility is not a large consideration, implementors may wish to use such grammars.

Some historical implementations have allowed some built-in functions to be called without an argument list, the result being a default argument list chosen in some “reasonable” way. Use of **length** as a synonym for **length(\$0)** is the only one of these forms that is thought to be widely known or widely used; this particular form is documented in various places (for example, most historical *awk* reference pages, although not in the referenced *The AWK Programming Language*) as legitimate practice. With this exception, default argument lists have always been undocumented and vaguely defined, and it is not at all clear how (or if) they should be generalized to user-defined functions. They add no useful functionality and preclude possible future extensions that might need to name functions without calling them. Not standardizing them seems the simplest course. The standard developers considered that **length** merited special treatment, however, since it has been documented in the past and sees possibly substantial use in historical programs. Accordingly, this usage has been made legitimate, but Issue 5 removed the obsolescent marking for XSI-conforming implementations and many otherwise conforming applications depend on this feature.

In **sub** and **gsub**, if *repl* is a string literal (the lexical token **STRING**), then two consecutive **<backslash>** characters should be used in the string to ensure a single **<backslash>** will precede the **<ampersand>** when the resultant string is passed to the function. (For example, to specify one literal **<ampersand>** in the replacement string, use **gsub(ERE, "\\&")**.)

Historically, the only special character in the *repl* argument of **sub** and **gsub** string functions was the **<ampersand>** (**'&'**) character and preceding it with the **<backslash>** character was used to turn off its special meaning.

The description in the ISO POSIX-2:1993 standard introduced behavior such that the **<backslash>** character was another special character and it was unspecified whether there were any other special characters. This description introduced several portability problems, some of which are described below, and so it has been replaced with the more historical description. Some of the problems include:

- Historically, to create the replacement string, a script could use `gsub(ERE, "\\&")`, but with the ISO POSIX-2:1993 standard wording, it was necessary to use `gsub(ERE, "\\\&")`. The `<backslash>` characters are doubled here because all string literals are subject to lexical analysis, which would reduce each pair of `<backslash>` characters to a single `<backslash>` before being passed to `gsub`.
- Since it was unspecified what the special characters were, for portable scripts to guarantee that characters are printed literally, each character had to be preceded with a `<backslash>`. (For example, a portable script had to use `gsub(ERE, "\\h\\i")` to produce a replacement string of "hi".)

The description for comparisons in the ISO POSIX-2:1993 standard did not properly describe historical practice because of the way numeric strings are compared as numbers. The current rules cause the following code:

```
if (0 == "000")
    print "strange, but true"
else
    print "not true"
```

to do a numeric comparison, causing the `if` to succeed. It should be intuitively obvious that this is incorrect behavior, and indeed, no historical implementation of *awk* actually behaves this way.

To fix this problem, the definition of *numeric string* was enhanced to include only those values obtained from specific circumstances (mostly external sources) where it is not possible to determine unambiguously whether the value is intended to be a string or a numeric.

Variables that are assigned to a numeric string shall also be treated as a numeric string. (For example, the notion of a numeric string can be propagated across assignments.) In comparisons, all variables having the uninitialized value are to be treated as a numeric operand evaluating to the numeric value zero.

Uninitialized variables include all types of variables including scalars, array elements, and fields. The definition of an uninitialized value in [Variables and Special Variables](#) (on page 2437) is necessary to describe the value placed on uninitialized variables and on fields that are valid (for example, `< $NF`) but have no characters in them and to describe how these variables are to be used in comparisons. A valid field, such as `$1`, that has no characters in it can be obtained from an input line of `"\t\t"` when `FS='\t'`. Historically, the comparison (`$1<10`) was done numerically after evaluating `$1` to the value zero.

The phrase "... also shall have the numeric value of the numeric string" was removed from several sections of the ISO POSIX-2:1993 standard because it specifies an unnecessary implementation detail. It is not necessary for POSIX.1-2008 to specify that these objects be assigned two different values. It is only necessary to specify that these objects may evaluate to two different values depending on context.

Historical implementations of *awk* did not parse hexadecimal integer or floating constants like `"0xa"` and `"0xap0"`. Due to an oversight, the 2001 through 2004 editions of this standard required support for hexadecimal floating constants. This was due to the reference to `atof()`. This version of the standard allows but does not require implementations to use `atof()` and includes a description of how floating-point numbers are recognized as an alternative to match historic behavior. The intent of this change is to allow implementations to recognize floating-point constants according to either the ISO/IEC 9899:1990 standard or ISO/IEC 9899:1999 standard, and to allow (but not require) implementations to recognize hexadecimal integer constants.

Historical implementations of *awk* did not support floating-point infinities and NaNs in *numeric*

strings; e.g., `"-INF"` and `"NaN"`. However, implementations that use the `atof()` or `strtod()` functions to do the conversion picked up support for these values if they used a ISO/IEC 9899:1999 standard version of the function instead of a ISO/IEC 9899:1990 standard version. Due to an oversight, the 2001 through 2004 editions of this standard did not allow support for infinities and NaNs, but in this revision support is allowed (but not required). This is a silent change to the behavior of *awk* programs; for example, in the POSIX locale the expression:

```
("-INF" + 0 < 0)
```

formerly had the value 0 because `"-INF"` converted to 0, but now it may have the value 0 or 1.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 1.3 (on page 2287), *grep*, *lex*, *sed*

XBD Chapter 5 (on page 121), Section 6.1 (on page 125), Chapter 8 (on page 173), Chapter 9 (on page 181), Section 12.2 (on page 215)

XSH *atof()*, *exec*, *isspace()*, *popen()*, *setlocale()*, *strtod()*

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

The *awk* utility is aligned with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term “must” for application requirements.

IEEE PASC Interpretation 1003.2 #211 is applied, adding the sentence “An occurrence of two consecutive `<backslash>` characters shall be interpreted as just a single literal `<backslash>` character.” into the description of the **sub** string function.

Issue 7

PASC Interpretation 1003.2-1992 #107 (SD5-XCU-ERN-73) is applied, updating the description of the **OFS** variable.

Austin Group Interpretation 1003.1-2001 #189 is applied.

Austin Group Interpretation 1003.1-2001 #201 is applied, permitting implementations to support infinities and NaNs.

SD5-XCU-ERN-79 is applied, restoring the horizontal lines to Table 4-1 (on page 2433), and SD5-XCU-ERN-80 is applied, changing the order of some table entries.

SD5-XCU-ERN-87 is applied, updating the descriptive text of the Grammar.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The EXTENDED DESCRIPTION is changed to make the support of hexadecimal integer and floating constants optional.

basename

Utilities

78578 **NAME**78579 **basename** — return non-directory portion of a pathname78580 **SYNOPSIS**78581 **basename** *string* [*suffix*]78582 **DESCRIPTION**

78583 The *string* operand shall be treated as a pathname, as defined in XBD [Section 3.266](#) (on page 75).
 78584 The string *string* shall be converted to the filename corresponding to the last pathname
 78585 component in *string* and then the suffix string *suffix*, if present, shall be removed. This shall be
 78586 done by performing actions equivalent to the following steps in order:

- 78587 1. If *string* is a null string, it is unspecified whether the resulting string is *.* or a null
 78588 string. In either case, skip steps 2 through 6.
- 78589 2. If *string* is *"/"*, it is implementation-defined whether steps 3 to 6 are skipped or
 78590 processed.
- 78591 3. If *string* consists entirely of *<slash>* characters, *string* shall be set to a single *<slash>*
 78592 character. In this case, skip steps 4 to 6.
- 78593 4. If there are any trailing *<slash>* characters in *string*, they shall be removed.
- 78594 5. If there are any *<slash>* characters remaining in *string*, the prefix of *string* up to and
 78595 including the last *<slash>* character in *string* shall be removed.
- 78596 6. If the *suffix* operand is present, is not identical to the characters remaining in *string*, and is
 78597 identical to a suffix of the characters remaining in *string*, the suffix *suffix* shall be removed
 78598 from *string*. Otherwise, *string* is not modified by this step. It shall not be considered an
 78599 error if *suffix* is not found in *string*.

78600 The resulting string shall be written to standard output.

78601 **OPTIONS**

78602 None.

78603 **OPERANDS**

78604 The following operands shall be supported:

78605 *string* A string.

78606 *suffix* A string.

78607 **STDIN**

78608 Not used.

78609 **INPUT FILES**

78610 None.

78611 **ENVIRONMENT VARIABLES**78612 The following environment variables shall affect the execution of *basename*:

78613 *LANG* Provide a default value for the internationalization variables that are unset or null.
 78614 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 78615 variables used to determine the values of locale categories.)

78616 *LC_ALL* If set to a non-empty string value, override the values of all the other
 78617 internationalization variables.

78618 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 78619 characters (for example, single-byte as opposed to multi-byte characters in
 78620 arguments).

Utilities

basename

78621 **LC_MESSAGES**
 78622 Determine the locale that should be used to affect the format and contents of
 78623 diagnostic messages written to standard error.

78624 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

78625 **ASYNCHRONOUS EVENTS**
 78626 Default.

78627 **STDOUT**
 78628 The *basename* utility shall write a line to the standard output in the following format:
 78629 "%s\n", <resulting string>

78630 **STDERR**
 78631 The standard error shall be used only for diagnostic messages.

78632 **OUTPUT FILES**
 78633 None.

78634 **EXTENDED DESCRIPTION**
 78635 None.

78636 **EXIT STATUS**
 78637 The following exit values shall be returned:
 78638 0 Successful completion.
 78639 >0 An error occurred.

78640 **CONSEQUENCES OF ERRORS**
 78641 Default.

78642 **APPLICATION USAGE**
 78643 The definition of *pathname* specifies implementation-defined behavior for pathnames starting
 78644 with two <slash> characters. Therefore, applications shall not arbitrarily add <slash> characters
 78645 to the beginning of a pathname unless they can ensure that there are more or less than two or are
 78646 prepared to deal with the implementation-defined consequences.

78647 **EXAMPLES**
 78648 If the string *string* is a valid pathname:
 78649 \$(basename "*string*")
 78650 produces a filename that could be used to open the file named by *string* in the directory returned
 78651 by:
 78652 \$(dirname "*string*")
 78653 If the string *string* is not a valid pathname, the same algorithm is used, but the result need not be
 78654 a valid filename. The *basename* utility is not expected to make any judgements about the validity
 78655 of *string* as a pathname; it just follows the specified algorithm to produce a result string.

78656 The following shell script compiles **/usr/src/cmd/cat.c** and moves the output to a file named **cat**
 78657 in the current directory when invoked with the argument **/usr/src/cmd/cat** or with the argument
 78658 **/usr/src/cmd/cat.c**:
 78659 c99 \$(dirname "\$1")/\$(basename "\$1" .c) .c
 78660 mv a.out \$(basename "\$1" .c)

basename

Utilities

78661 **RATIONALE**

78662 The behaviors of *basename* and *dirname* have been coordinated so that when *string* is a valid
78663 pathname:

78664 `$(basename "string")`

78665 would be a valid filename for the file in the directory:

78666 `$(dirname "string")`

78667 This would not work for the early proposal versions of these utilities due to the way it specified
78668 handling of trailing <slash> characters.

78669 Since the definition of *pathname* specifies implementation-defined behavior for pathnames
78670 starting with two <slash> characters, this volume of POSIX.1-2008 specifies similar
78671 implementation-defined behavior for the *basename* and *dirname* utilities.

78672 **FUTURE DIRECTIONS**

78673 None.

78674 **SEE ALSO**

78675 [Section 2.5](#) (on page 2301), *dirname*

78676 XBD [Section 3.266](#) (on page 75), [Chapter 8](#) (on page 173)

78677 **CHANGE HISTORY**

78678 First released in Issue 2.

78679 **Issue 6**

78680 IEEE PASC Interpretation 1003.2 #164 is applied.

78681 The normative text is reworded to avoid use of the term “must” for application requirements.

Utilities

batch

NAME

batch — schedule commands to be executed in a batch queue

SYNOPSIS

batch

DESCRIPTION

The *batch* utility shall read commands from standard input and schedule them for execution in a batch queue. It shall be the equivalent of the command:

```
at -q b -m now
```

where queue *b* is a special *at* queue, specifically for batch jobs. Batch jobs shall be submitted to the batch queue with no time constraints and shall be run by the system using algorithms, based on unspecified factors, that may vary with each invocation of *batch*.

XSI

Users shall be permitted to use *batch* if their name appears in the file **at.allow** which is located in an implementation-defined directory. If that file does not exist, the file **at.deny**, which is located in an implementation-defined directory, shall be checked to determine whether the user shall be denied access to *batch*. If neither file exists, only a process with appropriate privileges shall be allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted. The **at.allow** and **at.deny** files shall consist of one user name per line.

OPTIONS

None.

OPERANDS

None.

STDIN

The standard input shall be a text file consisting of commands acceptable to the shell command language described in [Chapter 2](#) (on page 2297).

INPUT FILES

XSI

The text files **at.allow** and **at.deny**, which are located in an implementation-defined directory, shall contain zero or more user names, one per line, of users who are, respectively, authorized or denied access to the *at* and *batch* utilities.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *batch*:

LANG Provide a default value for the internationalization variables that are unset or null. (See [XBD Section 8.2](#) (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

LC_TIME Determine the format and contents for date and time strings written by *batch*.

batch*Utilities*

78725	XSI	NLSPATH	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
78726		SHELL	Determine the name of a command interpreter to be used to invoke the at-job. If the variable is unset or null, <i>sh</i> shall be used. If it is set to a value other than a name for <i>sh</i> , the implementation shall do one of the following: use that shell; use <i>sh</i> ; use the login shell from the user database; any of the preceding accompanied by a warning diagnostic about which was chosen.
78727			
78728			
78729			
78730			
78731		TZ	Determine the timezone. The job shall be submitted for execution at the time specified by <i>timespec</i> or <i>-t time</i> relative to the timezone specified by the <i>TZ</i> variable. If <i>timespec</i> specifies a timezone, it overrides <i>TZ</i> . If <i>timespec</i> does not specify a timezone and <i>TZ</i> is unset or null, an unspecified default timezone shall be used.
78732			
78733			
78734			
78735			
78736		ASYNCHRONOUS EVENTS	
78737			Default.
78738		STDOUT	
78739			When standard input is a terminal, prompts of unspecified format for each line of the user input described in the STDIN section may be written to standard output.
78740			
78741		STDERR	
78742			The following shall be written to standard error when a job has been successfully submitted:
78743			"job %s at %s\n", <i>at_job_id</i> , <date>
78744			where <i>date</i> shall be equivalent in format to the output of:
78745			date +"%a %b %e %T %Y"
78746			The date and time written shall be adjusted so that they appear in the timezone of the user (as determined by the <i>TZ</i> variable).
78747			
78748			Neither this, nor warning messages concerning the selection of the command interpreter, are considered a diagnostic that changes the exit status.
78749			
78750			Diagnostic messages, if any, shall be written to standard error.
78751		OUTPUT FILES	
78752			None.
78753		EXTENDED DESCRIPTION	
78754			None.
78755		EXIT STATUS	
78756			The following exit values shall be returned:
78757			0 Successful completion.
78758			>0 An error occurred.
78759		CONSEQUENCES OF ERRORS	
78760			The job shall not be scheduled.

APPLICATION USAGE

It may be useful to redirect standard output within the specified commands.

EXAMPLES

1. This sequence can be used at a terminal:

```
batch
sort < file >outfile
EOT
```

2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a command procedure (the sequence of output redirection specifications is significant):

```
batch <<!
diff file1 file2 2>&1 >outfile | mailx mygroup
!
```

RATIONALE

Early proposals described *batch* in a manner totally separated from *at*, even though the historical model treated it almost as a synonym for *at -qb*. A number of features were added to list and control batch work separately from those in *at*. Upon further reflection, it was decided that the benefit of this did not merit the change to the historical interface.

The *-m* option was included on the equivalent *at* command because it is historical practice to mail results to the submitter, even if all job-produced output is redirected. As explained in the RATIONALE for *at*, the **now** keyword submits the job for immediate execution (after scheduling delays), despite some historical systems where *at now* would have been considered an error.

FUTURE DIRECTIONS

None.

SEE ALSO

at

XBD Chapter 8 (on page 173)

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The NAME is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term “must” for application requirements.

Issue 7

The *batch* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced by the *batch* utility.

bc

Utilities

78798 **NAME**78799 **bc** — arbitrary-precision arithmetic language78800 **SYNOPSIS**78801 **bc** [-l] [*file*...]78802 **DESCRIPTION**

78803 The *bc* utility shall implement an arbitrary precision calculator. It shall take input from any files
 78804 given, then read from the standard input. If the standard input and standard output to *bc* are
 78805 attached to a terminal, the invocation of *bc* shall be considered to be *interactive*, causing
 78806 behavioral constraints described in the following sections.

78807 **OPTIONS**78808 The *bc* utility shall conform to XBD Section 12.2 (on page 215).

78809 The following option shall be supported:

78810 **-l** (The letter ell.) Define the math functions and initialize *scale* to 20, instead of the
 78811 default zero; see the EXTENDED DESCRIPTION section.

78812 **OPERANDS**

78813 The following operand shall be supported:

78814 *file* A pathname of a text file containing *bc* program statements. After all *files* have
 78815 been read, *bc* shall read the standard input.

78816 **STDIN**

78817 See the INPUT FILES section.

78818 **INPUT FILES**

78819 Input files shall be text files containing a sequence of comments, statements, and function
 78820 definitions that shall be executed as they are read.

78821 **ENVIRONMENT VARIABLES**78822 The following environment variables shall affect the execution of *bc*:

78823 **LANG** Provide a default value for the internationalization variables that are unset or null.
 78824 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 78825 variables used to determine the values of locale categories.)

78826 **LC_ALL** If set to a non-empty string value, override the values of all the other
 78827 internationalization variables.

78828 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 78829 characters (for example, single-byte as opposed to multi-byte characters in
 78830 arguments and input files).

78831 **LC_MESSAGES**

78832 Determine the locale that should be used to affect the format and contents of
 78833 diagnostic messages written to standard error.

78834 **XSI NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

78835 **ASYNCHRONOUS EVENTS**

78836 Default.

78837 **STDOUT**

78838 The output of the *bc* utility shall be controlled by the program read, and consist of zero or more
 78839 lines containing the value of all executed expressions without assignments. The radix and
 78840 precision of the output shall be controlled by the values of the **obase** and **scale** variables; see the
 78841 EXTENDED DESCRIPTION section.

78842 **STDERR**

78843 The standard error shall be used only for diagnostic messages.

78844 **OUTPUT FILES**

78845 None.

78846 **EXTENDED DESCRIPTION**78847 **Grammar**

78848 The grammar in this section and the lexical conventions in the following section shall together
 78849 describe the syntax for *bc* programs. The general conventions for this style of grammar are
 78850 described in [Section 1.3](#) (on page 2287). A valid program can be represented as the non-terminal
 78851 symbol **program** in the grammar. This formal syntax shall take precedence over the text syntax
 78852 description.

78853	%token	EOF NEWLINE STRING LETTER NUMBER	
78854	%token	MUL_OP	
78855	/*	'*', '/', '%'	*/
78856	%token	ASSIGN_OP	
78857	/*	'=', '+=', '==', '*=', '/=', '%=', '^='	*/
78858	%token	REL_OP	
78859	/*	'==', '<=', '>=', '!=', '<', '>'	*/
78860	%token	INCR_DECR	
78861	/*	'++', '--'	*/
78862	%token	Define Break Quit Length	
78863	/*	'define', 'break', 'quit', 'length'	*/
78864	%token	Return For If While Sqrt	
78865	/*	'return', 'for', 'if', 'while', 'sqrt'	*/
78866	%token	Scale Ibase Obase Auto	
78867	/*	'scale', 'ibase', 'obase', 'auto'	*/
78868	%start	program	
78869	%%		
78870	program	:	EOF
78871			input_item program
78872		;	
78873	input_item	:	semicolon_list NEWLINE
78874			function
78875		;	
78876	semicolon_list	:	/* empty */
78877			statement
78878			semicolon_list ';' statement
78879			semicolon_list ';' ;
78880		;	
78881	statement_list	:	/* empty */
78882			statement
78883			statement_list NEWLINE

```

78884      | statement_list NEWLINE statement
78885      | statement_list ';'
78886      | statement_list ';' statement
78887      ;
78888      statement      : expression
78889      | STRING
78890      | Break
78891      | Quit
78892      | Return
78893      | Return '(' return_expression ')'
78894      | For '(' expression ';'
78895      |     relational_expression ';'
78896      |     expression ')' statement
78897      | If '(' relational_expression ')' statement
78898      | While '(' relational_expression ')' statement
78899      | '{' statement_list '}'
78900      ;
78901      function      : Define LETTER '(' opt_parameter_list ')'
78902      |     '{' NEWLINE opt_auto_define_list
78903      |     statement_list '}'
78904      ;
78905      opt_parameter_list : /* empty */
78906      | parameter_list
78907      ;
78908      parameter_list   : LETTER
78909      | define_list ',' LETTER
78910      ;
78911      opt_auto_define_list : /* empty */
78912      | Auto define_list NEWLINE
78913      | Auto define_list ';'
78914      ;
78915      define_list      : LETTER
78916      | LETTER '[' ']'
78917      | define_list ',' LETTER
78918      | define_list ',' LETTER '[' ']'
78919      ;
78920      opt_argument_list : /* empty */
78921      | argument_list
78922      ;
78923      argument_list    : expression
78924      | LETTER '[' ']' ',' argument_list
78925      ;
78926      relational_expression : expression
78927      | expression REL_OP expression
78928      ;
78929      return_expression  : /* empty */
78930      | expression

```

```

78931                                     ;
78932     expression                       : named_expression
78933                                     | NUMBER
78934                                     | '(' expression ')'
78935                                     | LETTER '(' opt_argument_list ')'
78936                                     | '-' expression
78937                                     | expression '+' expression
78938                                     | expression '-' expression
78939                                     | expression MUL_OP expression
78940                                     | expression '^' expression
78941                                     | INCR_DECR named_expression
78942                                     | named_expression INCR_DECR
78943                                     | named_expression ASSIGN_OP expression
78944                                     | Length '(' expression ')'
78945                                     | Sqrt '(' expression ')'
78946                                     | Scale '(' expression ')'
78947                                     ;
78948     named_expression                 : LETTER
78949                                     | LETTER '[' expression ']'
78950                                     | Scale
78951                                     | Ibase
78952                                     | Obase
78953                                     ;

```

Lexical Conventions in bc

The lexical conventions for *bc* programs, with respect to the preceding grammar, shall be as follows:

1. Except as noted, *bc* shall recognize the longest possible token or delimiter beginning at a given point.
2. A comment shall consist of any characters beginning with the two adjacent characters `"/*"` and terminated by the next occurrence of the two adjacent characters `"*/"`. Comments shall have no effect except to delimit lexical tokens.
3. The `<newline>` shall be recognized as the token **NEWLINE**.
4. The token **STRING** shall represent a string constant; it shall consist of any characters beginning with the double-quote character (`'"`) and terminated by another occurrence of the double-quote character. The value of the string is the sequence of all characters between, but not including, the two double-quote characters. All characters shall be taken literally from the input, and there is no way to specify a string containing a double-quote character. The length of the value of each string shall be limited to `{BC_STRING_MAX}` bytes.
5. A `<blank>` shall have no effect except as an ordinary character if it appears within a **STRING** token, or to delimit a lexical token other than **STRING**.
6. The combination of a `<backslash>` character immediately followed by a `<newline>` shall have no effect other than to delimit lexical tokens with the following exceptions:

- 78974 • It shall be interpreted as the character sequence "<newline>" in **STRING** tokens.
- 78975 • It shall be ignored as part of a multi-line **NUMBER** token.
- 78976 7. The token **NUMBER** shall represent a numeric constant. It shall be recognized by the
- 78977 following grammar:
- 78978 NUMBER : integer
- 78979 | '.' integer
- 78980 | integer '.'
- 78981 | integer '.' integer
- 78982 ;
- 78983 integer : digit
- 78984 | integer digit
- 78985 ;
- 78986 digit : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
- 78987 | 8 | 9 | A | B | C | D | E | F
- 78988 ;
- 78989 8. The value of a **NUMBER** token shall be interpreted as a numeral in the base specified by
- 78990 the value of the internal register **ibase** (described below). Each of the **digit** characters
- 78991 shall have the value from 0 to 15 in the order listed here, and the <period> character shall
- 78992 represent the radix point. The behavior is undefined if digits greater than or equal to the
- 78993 value of **ibase** appear in the token. However, note the exception for single-digit values
- 78994 being assigned to **ibase** and **obase** themselves, in [Operations in bc](#) (on page 2475).
- 78995 9. The following keywords shall be recognized as tokens:
- 78996 **auto** **ibase** **length** **return** **while**
- 78997 **break** **if** **obase** **scale**
- 78998 **define** **for** **quit** **sqrt**
- 78999 10. Any of the following characters occurring anywhere except within a keyword shall be
- 79000 recognized as the token **LETTER**:
- 79001 a b c d e f . g h i j k l m n o p q r s t u v w x y z
- 79002 11. The following single-character and two-character sequences shall be recognized as the
- 79003 token **ASSIGN_OP**:
- 79004 = += -= *= /= %= ^=
- 79005 12. If an '=' character, as the beginning of a token, is followed by a '-' character with no
- 79006 intervening delimiter, the behavior is undefined.
- 79007 13. The following single-characters shall be recognized as the token **MUL_OP**:
- 79008 * / %
- 79009 14. The following single-character and two-character sequences shall be recognized as the
- 79010 token **REL_OP**:
- 79011 == <= >= != < >
- 79012 15. The following two-character sequences shall be recognized as the token **INCR_DECR**:
- 79013 ++ --

16. The following single characters shall be recognized as tokens whose names are the character:

<newline> () , + - ; [] ^ { }

17. The token **EOF** is returned when the end of input is reached.

Operations in bc

There are three kinds of identifiers: ordinary identifiers, array identifiers, and function identifiers. All three types consist of single lowercase letters. Array identifiers shall be followed by square brackets ("[]"). An array subscript is required except in an argument or auto list. Arrays are singly dimensioned and can contain up to {BC_DIM_MAX} elements. Indexing shall begin at zero so an array is indexed from 0 to {BC_DIM_MAX}-1. Subscripts shall be truncated to integers. The application shall ensure that function identifiers are followed by parentheses, possibly enclosing arguments. The three types of identifiers do not conflict.

The following table summarizes the rules for precedence and associativity of all operators. Operators on the same line shall have the same precedence; rows are in order of decreasing precedence.

Table 4-3 Operators in bc

Operator	Associativity
++, --	N/A
unary -	N/A
^	Right to left
*, /, %	Left to right
+, binary -	Left to right
=, +=, -=, *=, /=, %=, ^=	Right to left
==, <=, >=, !=, <, >	None

Each expression or named expression has a *scale*, which is the number of decimal digits that shall be maintained as the fractional portion of the expression.

Named expressions are places where values are stored. Named expressions shall be valid on the left side of an assignment. The value of a named expression shall be the value stored in the place named. Simple identifiers and array elements are named expressions; they have an initial value of zero and an initial scale of zero.

The internal registers **scale**, **ibase**, and **obase** are all named expressions. The scale of an expression consisting of the name of one of these registers shall be zero; values assigned to any of these registers are truncated to integers. The **scale** register shall contain a global value used in computing the scale of expressions (as described below). The value of the register **scale** is limited to $0 \leq \text{scale} \leq \{\text{BC\_SCALE\_MAX}\}$ and shall have a default value of zero. The **ibase** and **obase** registers are the input and output number radix, respectively. The value of **ibase** shall be limited to:

$$2 \leq \text{ibase} \leq 16$$

The value of **obase** shall be limited to:

$$2 \leq \text{obase} \leq \{\text{BC\_BASE\_MAX}\}$$

When either **ibase** or **obase** is assigned a single **digit** value from the list in [Lexical Conventions in bc](#) (on page 2473), the value shall be assumed in hexadecimal. (For example, **ibase=A** sets to base ten, regardless of the current **ibase** value.) Otherwise, the behavior is undefined when

79057 digits greater than or equal to the value of **ibase** appear in the input. Both **ibase** and **obase** shall
 79058 have initial values of 10.

79059 Internal computations shall be conducted as if in decimal, regardless of the input and output
 79060 bases, to the specified number of decimal digits. When an exact result is not achieved (for
 79061 example, **scale**=0; 3.2/1), the result shall be truncated.

79062 For all values of **obase** specified by this volume of POSIX.1-2008, *bc* shall output numeric values
 79063 by performing each of the following steps in order:

- 79064 1. If the value is less than zero, a <hyphen> (‘-’) character shall be output.
- 79065 2. One of the following is output, depending on the numerical value:
 - 79066 • If the absolute value of the numerical value is greater than or equal to one, the
 79067 integer portion of the value shall be output as a series of digits appropriate to **obase**
 79068 (as described below), most significant digit first. The most significant non-zero digit
 79069 shall be output next, followed by each successively less significant digit.
 - 79070 • If the absolute value of the numerical value is less than one but greater than zero
 79071 and the scale of the numerical value is greater than zero, it is unspecified whether
 79072 the character 0 is output.
 - 79073 • If the numerical value is zero, the character 0 shall be output.
- 79074 3. If the scale of the value is greater than zero and the numeric value is not zero, a <period>
 79075 character shall be output, followed by a series of digits appropriate to **obase** (as described
 79076 below) representing the most significant portion of the fractional part of the value. If *s*
 79077 represents the scale of the value being output, the number of digits output shall be *s* if
 79078 **obase** is 10, less than or equal to *s* if **obase** is greater than 10, or greater than or equal to *s*
 79079 if **obase** is less than 10. For **obase** values other than 10, this should be the number of
 79080 digits needed to represent a precision of 10^s .

79081 For **obase** values from 2 to 16, valid digits are the first **obase** of the single characters:

79082 0 1 2 3 4 5 6 7 8 9 A B C D E F

79083 which represent the values zero to 15, inclusive, respectively.

79084 For bases greater than 16, each digit shall be written as a separate multi-digit decimal number.
 79085 Each digit except the most significant fractional digit shall be preceded by a single <space>. For
 79086 bases from 17 to 100, *bc* shall write two-digit decimal numbers; for bases from 101 to 1 000, three-
 79087 digit decimal strings, and so on. For example, the decimal number 1 024 in base 25 would be
 79088 written as:

79089 Δ01Δ15Δ24

79090 and in base 125, as:

79091 Δ008Δ024

79092 Very large numbers shall be split across lines with 70 characters per line in the POSIX locale;
 79093 other locales may split at different character boundaries. Lines that are continued shall end with
 79094 a <backslash>.

79095 A function call shall consist of a function name followed by parentheses containing a
 79096 <comma>-separated list of expressions, which are the function arguments. A whole array
 79097 passed as an argument shall be specified by the array name followed by empty square brackets.
 79098 All function arguments shall be passed by value. As a result, changes made to the formal
 79099 parameters shall have no effect on the actual arguments. If the function terminates by executing

79100 a **return** statement, the value of the function shall be the value of the expression in the
 79101 parentheses of the **return** statement or shall be zero if no expression is provided or if there is no
 79102 **return** statement.

79103 The result of **sqrt**(*expression*) shall be the square root of the expression. The result shall be
 79104 truncated in the least significant decimal place. The scale of the result shall be the scale of the
 79105 expression or the value of **scale**, whichever is larger.

79106 The result of **length**(*expression*) shall be the total number of significant decimal digits in the
 79107 expression. The scale of the result shall be zero.

79108 The result of **scale**(*expression*) shall be the scale of the expression. The scale of the result shall be
 79109 zero.

79110 A numeric constant shall be an expression. The scale shall be the number of digits that follow the
 79111 radix point in the input representing the constant, or zero if no radix point appears.

79112 The sequence (*expression*) shall be an expression with the same value and scale as *expression*.
 79113 The parentheses can be used to alter the normal precedence.

79114 The semantics of the unary and binary operators are as follows:

79115 *-expression*
 79116 The result shall be the negative of the *expression*. The scale of the result shall be the scale of
 79117 *expression*.

79118 The unary increment and decrement operators shall not modify the scale of the named
 79119 expression upon which they operate. The scale of the result shall be the scale of that named
 79120 expression.

79121 *++named-expression*
 79122 The named expression shall be incremented by one. The result shall be the value of the
 79123 named expression after incrementing.

79124 *--named-expression*
 79125 The named expression shall be decremented by one. The result shall be the value of the
 79126 named expression after decrementing.

79127 *named-expression++*
 79128 The named expression shall be incremented by one. The result shall be the value of the
 79129 named expression before incrementing.

79130 *named-expression--*
 79131 The named expression shall be decremented by one. The result shall be the value of the
 79132 named expression before decrementing.

79133 The exponentiation operator, <circumflex> (' ^ '), shall bind right to left.

79134 *expression^expression*
 79135 The result shall be the first *expression* raised to the power of the second *expression*. If the
 79136 second expression is not an integer, the behavior is undefined. If *a* is the scale of the left
 79137 expression and *b* is the absolute value of the right expression, the scale of the result shall be:
 79138 if $b \geq 0$ $\min(a * b, \max(\text{scale}, a))$ if $b < 0$ *scale*

79139 The multiplicative operators (' * ' , ' / ' , ' % ') shall bind left to right.

79140 *expression*expression*
 79141 The result shall be the product of the two expressions. If *a* and *b* are the scales of the two
 79142 expressions, then the scale of the result shall be:

79143 $\min(a+b, \max(\text{scale}, a, b))$

79144 *expression*/*expression*

79145 The result shall be the quotient of the two expressions. The scale of the result shall be the
79146 value of **scale**.

79147 *expression*%*expression*

79148 For expressions *a* and *b*, *a*%*b* shall be evaluated equivalent to the steps:

79149 1. Compute *a*/*b* to current scale.

79150 2. Use the result to compute:

79151 $a - (a / b) * b$

79152 to scale:

79153 $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

79154 The scale of the result shall be:

79155 $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

79156 When **scale** is zero, the '%' operator is the mathematical remainder operator.

79157 The additive operators ('+', '-') shall bind left to right.

79158 *expression*+*expression*

79159 The result shall be the sum of the two expressions. The scale of the result shall be the
79160 maximum of the scales of the expressions.

79161 *expression*-*expression*

79162 The result shall be the difference of the two expressions. The scale of the result shall be the
79163 maximum of the scales of the expressions.

79164 The assignment operators ('=', '+=', '-=', '*=', '/=', '%=', '^=') shall bind right to left.

79165 *named-expression*=*expression*

79166 This expression shall result in assigning the value of the expression on the right to the
79167 named expression on the left. The scale of both the named expression and the result shall be
79168 the scale of *expression*.

79169 The compound assignment forms:

79170 *named-expression* <operator>= *expression*

79171 shall be equivalent to:

79172 *named-expression*=*named-expression* <operator> *expression*

79173 except that the *named-expression* shall be evaluated only once.

79174 Unlike all other operators, the relational operators ('<', '>', '<=', '>=', '==', '!=') shall be
79175 only valid as the object of an **if**, **while**, or inside a **for** statement.

79176 *expression1*<*expression2*

79177 The relation shall be true if the value of *expression1* is strictly less than the value of
79178 *expression2*.

79179 *expression1*>*expression2*

79180 The relation shall be true if the value of *expression1* is strictly greater than the value of
79181 *expression2*.

Utilities

bc

expression1 <=*expression2*

The relation shall be true if the value of *expression1* is less than or equal to the value of *expression2*.

expression1 >=*expression2*

The relation shall be true if the value of *expression1* is greater than or equal to the value of *expression2*.

expression1 = *expression2*

The relation shall be true if the values of *expression1* and *expression2* are equal.

expression1 != *expression2*

The relation shall be true if the values of *expression1* and *expression2* are unequal.

There are only two storage classes in *bc*: global and automatic (local). Only identifiers that are local to a function need be declared with the **auto** command. The arguments to a function shall be local to the function. All other identifiers are assumed to be global and available to all functions. All identifiers, global and local, have initial values of zero. Identifiers declared as auto shall be allocated on entry to the function and released on returning from the function. They therefore do not retain values between function calls. Auto arrays shall be specified by the array name followed by empty square brackets. On entry to a function, the old values of the names that appear as parameters and as automatic variables shall be pushed onto a stack. Until the function returns, reference to these names shall refer only to the new values.

References to any of these names from other functions that are called from this function also refer to the new value until one of those functions uses the same name for a local variable.

When a statement is an expression, unless the main operator is an assignment, execution of the statement shall write the value of the expression followed by a <newline>.

When a statement is a string, execution of the statement shall write the value of the string.

Statements separated by <semicolon> or <newline> characters shall be executed sequentially. In an interactive invocation of *bc*, each time a <newline> is read that satisfies the grammatical production:

```
input_item : semicolon_list NEWLINE
```

the sequential list of statements making up the **semicolon_list** shall be executed immediately and any output produced by that execution shall be written without any delay due to buffering.

In an **if** statement (**if**(*relation*) *statement*), the *statement* shall be executed if the relation is true.

The **while** statement (**while**(*relation*) *statement*) implements a loop in which the *relation* is tested; each time the *relation* is true, the *statement* shall be executed and the *relation* retested. When the *relation* is false, execution shall resume after *statement*.

A **for** statement (**for**(*expression*; *relation*; *expression*) *statement*) shall be the same as:

```
first-expression
while (relation) {
    statement
    last-expression
}
```

The application shall ensure that all three expressions are present.

The **break** statement shall cause termination of a **for** or **while** statement.

The **auto** statement (**auto** *identifier* [*identifier*] ...) shall cause the values of the identifiers to be

pushed down. The identifiers can be ordinary identifiers or array identifiers. Array identifiers shall be specified by following the array name by empty square brackets. The application shall ensure that the **auto** statement is the first statement in a function definition.

A **define** statement:

```
define LETTER ( opt_parameter_list ) {
    opt_auto_define_list
    statement_list
}
```

defines a function named **LETTER**. If a function named **LETTER** was previously defined, the **define** statement shall replace the previous definition. The expression:

```
LETTER ( opt_argument_list )
```

shall invoke the function named **LETTER**. The behavior is undefined if the number of arguments in the invocation does not match the number of parameters in the definition. Functions shall be defined before they are invoked. A function shall be considered to be defined within its own body, so recursive calls are valid. The values of numeric constants within a function shall be interpreted in the base specified by the value of the **ibase** register when the function is invoked.

The **return** statements (**return** and **return(expression)**) shall cause termination of a function, popping of its auto variables, and specification of the result of the function. The first form shall be equivalent to **return(0)**. The value and scale of the result returned by the function shall be the value and scale of the expression returned.

The **quit** statement (**quit**) shall stop execution of a *bc* program at the point where the statement occurs in the input, even if it occurs in a function definition, or in an **if**, **for**, or **while** statement.

The following functions shall be defined when the **-l** option is specified:

```
s( expression )
    Sine of argument in radians.

c( expression )
    Cosine of argument in radians.

a( expression )
    Arctangent of argument.

l( expression )
    Natural logarithm of argument.

e( expression )
    Exponential function of argument.

j( expression, expression )
    Bessel function of integer order.
```

The scale of the result returned by these functions shall be the value of the **scale** register at the time the function is invoked. The value of the **scale** register after these functions have completed their execution shall be the same value it had upon invocation. The behavior is undefined if any of these functions is invoked with an argument outside the domain of the mathematical function.

EXIT STATUS

The following exit values shall be returned:

0 All input files were processed successfully.

unspecified An error occurred.

CONSEQUENCES OF ERRORS

If any *file* operand is specified and the named file cannot be accessed, *bc* shall write a diagnostic message to standard error and terminate without any further action.

In an interactive invocation of *bc*, the utility should print an error message and recover following any error in the input. In a non-interactive invocation of *bc*, invalid input causes undefined behavior.

APPLICATION USAGE

Automatic variables in *bc* do not work in exactly the same way as in either C or PL/1.

For historical reasons, the exit status from *bc* cannot be relied upon to indicate that an error has occurred. Returning zero after an error is possible. Therefore, *bc* should be used primarily by interactive users (who can react to error messages) or by application programs that can somehow validate the answers returned as not including error messages.

The *bc* utility always uses the <period> (' . ') character to represent a radix point, regardless of any decimal-point character specified as part of the current locale. In languages like C or *awk*, the <period> character is used in program source, so it can be portable and unambiguous, while the locale-specific character is used in input and output. Because there is no distinction between source and input in *bc*, this arrangement would not be possible. Using the locale-specific character in *bc*'s input would introduce ambiguities into the language; consider the following example in a locale with a <comma> as the decimal-point character:

```
define f(a,b) {
    ...
}
...
f(1,2,3)
```

Because of such ambiguities, the <period> character is used in input. Having input follow different conventions from output would be confusing in either pipeline usage or interactive usage, so the <period> is also used in output.

EXAMPLES

In the shell, the following assigns an approximation of the first ten digits of ' π ' to the variable *x*:

```
x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

The following *bc* program prints the same approximation of ' π ', with a label, to standard output:

```
scale = 10
"pi equals "
104348 / 33215
```

The following defines a function to compute an approximate value of the exponential function (note that such a function is predefined if the *-l* option is specified):

```
scale = 20
define e(x) {
    auto a, b, c, i, s
```

```

79310         a = 1
79311         b = 1
79312         s = 1
79313         for (i = 1; 1 == 1; i++){
79314             a = a*x
79315             b = b*i
79316             c = a/b
79317             if (c == 0) {
79318                 return(s)
79319             }
79320             s = s+c
79321         }
79322     }

```

79323 The following prints approximate values of the exponential function of the first ten integers:

```

79324     for (i = 1; i <= 10; ++i) {
79325         e(i)
79326     }

```

79327 RATIONALE

79328 The *bc* utility is implemented historically as a front-end processor for *dc*; *dc* was not selected to
 79329 be part of this volume of POSIX.1-2008 because *bc* was thought to have a more intuitive
 79330 programmatic interface. Current implementations that implement *bc* using *dc* are expected to be
 79331 compliant.

79332 The exit status for error conditions has been left unspecified for several reasons:

- 79333 • The *bc* utility is used in both interactive and non-interactive situations. Different exit codes
 79334 may be appropriate for the two uses.
- 79335 • It is unclear when a non-zero exit should be given; divide-by-zero, undefined functions,
 79336 and syntax errors are all possibilities.
- 79337 • It is not clear what utility the exit status has.
- 79338 • In the 4.3 BSD, System V, and Ninth Edition implementations, *bc* works in conjunction with
 79339 *dc*. The *dc* utility is the parent, *bc* is the child. This was done to cleanly terminate *bc* if *dc*
 79340 aborted.

79341 The decision to have *bc* exit upon encountering an inaccessible input file is based on the belief
 79342 that *bc file1 file2* is used most often when at least *file1* contains data/function
 79343 declarations/initializations. Having *bc* continue with prerequisite files missing is probably not
 79344 useful. There is no implication in the CONSEQUENCES OF ERRORS section that *bc* must check
 79345 all its files for accessibility before opening any of them.

79346 There was considerable debate on the appropriateness of the language accepted by *bc*. Several
 79347 reviewers preferred to see either a pure subset of the C language or some changes to make the
 79348 language more compatible with C. While the *bc* language has some obvious similarities to C, it
 79349 has never claimed to be compatible with any version of C. An interpreter for a subset of C might
 79350 be a very worthwhile utility, and it could potentially make *bc* obsolete. However, no such utility
 79351 is known in historical practice, and it was not within the scope of this volume of POSIX.1-2008 to
 79352 define such a language and utility. If and when they are defined, it may be appropriate to
 79353 include them in a future version of this standard. This left the following alternatives:

1. Exclude any calculator language from this volume of POSIX.1-2008.

The consensus of the standard developers was that a simple programmatic calculator language is very useful for both applications and interactive users. The only arguments for excluding any calculator were that it would become obsolete if and when a C-compatible one emerged, or that the absence would encourage the development of such a C-compatible one. These arguments did not sufficiently address the needs of current application developers.

2. Standardize the historical *dc*, possibly with minor modifications.

The consensus of the standard developers was that *dc* is a fundamentally less usable language and that that would be far too severe a penalty for avoiding the issue of being similar to but incompatible with C.

3. Standardize the historical *bc*, possibly with minor modifications.

This was the approach taken. Most of the proponents of changing the language would not have been satisfied until most or all of the incompatibilities with C were resolved. Since most of the changes considered most desirable would break historical applications and require significant modification to historical implementations, almost no modifications were made. The one significant modification that was made was the replacement of the historical *bc* assignment operators "*=*", and so on, with the more modern "*+=*", and so on. The older versions are considered to be fundamentally flawed because of the lexical ambiguity in uses like *a=-1*.

In order to permit implementations to deal with backwards-compatibility as they see fit, the behavior of this one ambiguous construct was made undefined. (At least three implementations have been known to support this change already, so the degree of change involved should not be great.)

The '*%*' operator is the mathematical remainder operator when **scale** is zero. The behavior of this operator for other values of **scale** is from historical implementations of *bc*, and has been maintained for the sake of historical applications despite its non-intuitive nature.

Historical implementations permit setting **ibase** and **obase** to a broader range of values. This includes values less than 2, which were not seen as sufficiently useful to standardize. These implementations do not interpret input properly for values of **ibase** that are greater than 16. This is because numeric constants are recognized syntactically, rather than lexically, as described in this volume of POSIX.1-2008. They are built from lexical tokens of single hexadecimal digits and <period> characters. Since <blank> characters between tokens are not visible at the syntactic level, it is not possible to recognize the multi-digit "digits" used in the higher bases properly. The ability to recognize input in these bases was not considered useful enough to require modifying these implementations. Note that the recognition of numeric constants at the syntactic level is not a problem with conformance to this volume of POSIX.1-2008, as it does not impact the behavior of conforming applications (and correct *bc* programs). Historical implementations also accept input with all of the digits '*0*'–'*9*' and '*A*'–'*F*' regardless of the value of **ibase**; since digits with value greater than or equal to **ibase** are not really appropriate, the behavior when they appear is undefined, except for the common case of:

```
ibase=8;
/* Process in octal base. */
...
ibase=A
/* Restore decimal base. */
```

In some historical implementations, if the expression to be written is an uninitialized array

79401 element, a leading <space> and/or up to four leading 0 characters may be output before the
 79402 character zero. This behavior is considered a bug; it is unlikely that any currently conforming
 79403 application relies on:

79404 `echo 'b[3]' | bc`

79405 returning 00000 rather than 0.

79406 Exact calculation of the number of fractional digits to output for a given value in a base other
 79407 than 10 can be computationally expensive. Historical implementations use a faster
 79408 approximation, and this is permitted. Note that the requirements apply only to values of **obase**
 79409 that this volume of POSIX.1-2008 requires implementations to support (in particular, not to 1, 0,
 79410 or negative bases, if an implementation supports them as an extension).

79411 Historical implementations of *bc* did not allow array parameters to be passed as the last
 79412 parameter to a function. New implementations are encouraged to remove this restriction even
 79413 though it is not required by the grammar.

79414 FUTURE DIRECTIONS

79415 None.

79416 SEE ALSO

79417 [Section 1.3](#) (on page 2287), *awk*

79418 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

79419 CHANGE HISTORY

79420 First released in Issue 4.

79421 Issue 5

79422 The FUTURE DIRECTIONS section is added.

79423 Issue 6

79424 Updated to align with the IEEE P1003.2b draft standard, which included resolution of several
 79425 interpretations of the ISO POSIX-2:1993 standard.

79426 The normative text is reworded to avoid use of the term “must” for application requirements.

79427 Issue 7

79428 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Utilities

bg79429 **NAME**79430 **bg** — run jobs in the background79431 **SYNOPSIS**79432 UP **bg** [*job_id*...]79433 **DESCRIPTION**

79434 If job control is enabled (see the description of *set -m*), the *bg* utility shall resume suspended jobs
 79435 from the current environment (see [Section 2.12](#), on page 2331) by running them as background
 79436 jobs. If the job specified by *job_id* is already a running background job, the *bg* utility shall have
 79437 no effect and shall exit successfully.

79438 Using *bg* to place a job into the background shall cause its process ID to become “known in the
 79439 current shell execution environment”, as if it had been started as an asynchronous list; see
 79440 [Section 2.9.3.1](#) (on page 2319).

79441 **OPTIONS**

79442 None.

79443 **OPERANDS**

79444 The following operand shall be supported:

79445 *job_id* Specify the job to be resumed as a background job. If no *job_id* operand is given,
 79446 the most recently suspended job shall be used. The format of *job_id* is described in
 79447 XBD [Section 3.203](#) (on page 65).

79448 **STDIN**

79449 Not used.

79450 **INPUT FILES**

79451 None.

79452 **ENVIRONMENT VARIABLES**79453 The following environment variables shall affect the execution of *bg*:

79454 *LANG* Provide a default value for the internationalization variables that are unset or null.
 79455 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 79456 variables used to determine the values of locale categories.)

79457 *LC_ALL* If set to a non-empty string value, override the values of all the other
 79458 internationalization variables.

79459 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 79460 characters (for example, single-byte as opposed to multi-byte characters in
 79461 arguments).

79462 *LC_MESSAGES*

79463 Determine the locale that should be used to affect the format and contents of
 79464 diagnostic messages written to standard error.

79465 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

79466 **ASYNCHRONOUS EVENTS**

79467 Default.

79468 **STDOUT**79469 The output of *bg* shall consist of a line in the format:79470 "[*%d*] *%s*\n", <*job-number*>, <*command*>

79471 where the fields are as follows:

79472 <job-number> A number that can be used to identify the job to the *wait*, *fg*, and *kill* utilities. Using
79473 these utilities, the job can be identified by prefixing the job number with ' % '.

79474 <command> The associated command that was given to the shell.

79475 **STDERR**

79476 The standard error shall be used only for diagnostic messages.

79477 **OUTPUT FILES**

79478 None.

79479 **EXTENDED DESCRIPTION**

79480 None.

79481 **EXIT STATUS**

79482 The following exit values shall be returned:

79483 0 Successful completion.

79484 >0 An error occurred.

79485 **CONSEQUENCES OF ERRORS**

79486 If job control is disabled, the *bg* utility shall exit with an error and no job shall be placed in the
79487 background.

79488 **APPLICATION USAGE**

79489 A job is generally suspended by typing the SUSP character (<control>-Z on most systems); see
79490 XBD Chapter 11 (on page 199). At that point, *bg* can put the job into the background. This is
79491 most effective when the job is expecting no terminal input and its output has been redirected to
79492 non-terminal files. A background job can be forced to stop when it has terminal output by
79493 issuing the command:

79494 `stty tostop`

79495 A background job can be stopped with the command:

79496 `kill -s stop job ID`

79497 The *bg* utility does not work as expected when it is operating in its own utility execution
79498 environment because that environment has no suspended jobs. In the following examples:

79499 `... | xargs bg`
79500 `(bg)`

79501 each *bg* operates in a different environment and does not share its parent shell's understanding
79502 of jobs. For this reason, *bg* is generally implemented as a shell regular built-in.

79503 **EXAMPLES**

79504 None.

79505 **RATIONALE**

79506 The extensions to the shell specified in this volume of POSIX.1-2008 have mostly been based on
79507 features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also
79508 based on the KornShell. The standard developers examined the characteristics of the C shell
79509 versions of these utilities and found that differences exist. Despite widespread use of the C shell,
79510 the KornShell versions were selected for this volume of POSIX.1-2008 to maintain a degree of
79511 uniformity with the rest of the KornShell features selected (such as the very popular command
79512 line editing features).

79513 The *bg* utility is expected to wrap its output if the output exceeds the number of display
79514 columns.

79515 **FUTURE DIRECTIONS**

79516 None.

79517 **SEE ALSO**

79518 [Section 2.9.3.1](#) (on page 2319), *fg*, *kill*, *jobs*, *wait*

79519 XBD [Section 3.203](#) (on page 65), [Chapter 8](#) (on page 173), [Chapter 11](#) (on page 199)

79520 **CHANGE HISTORY**

79521 First released in Issue 4.

79522 **Issue 6**

79523 This utility is marked as part of the User Portability Utilities option.

79524 The JC margin marker on the SYNOPSIS is removed since support for Job Control is mandatory
79525 in this version. This is a FIPS requirement.

79526 **Issue 7**

79527 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

79528 NAME

79529 c99 — compile standard C programs

79530 SYNOPSIS

```
79531 CD c99 [options...] pathname [[pathname] [-I directory]
79532 [-L directory] [-l library]]...
```

79533 DESCRIPTION

79534 The *c99* utility is an interface to the standard C compilation system; it shall accept source code
 79535 conforming to the ISO C standard. The system conceptually consists of a compiler and link
 79536 editor. The input files referenced by *pathname* operands and *-I* option-arguments shall be
 79537 compiled and linked to produce an executable file. (It is unspecified whether the linking occurs
 79538 entirely within the operation of *c99*; some implementations may produce objects that are not
 79539 fully resolved until the file is executed.)

79540 If the *-c* option is specified, for all *pathname* operands of the form *file.c* the files:

79541 \$(basename *pathname* .c) .o

79542 shall be created as the result of successful compilation. If the *-c* option is not specified, it is
 79543 unspecified whether such .o files are created or deleted for the *file.c* operands.

79544 If there are no options that prevent link editing (such as *-c* or *-E*), and all input files compile and
 79545 link without error, the resulting executable file shall be written according to the *-o outfile* option
 79546 (if present) or to the file **a.out**.

79547 The executable file shall be created as specified in [Section 1.1.1.4](#) (on page 2280), except that the
 79548 file permission bits shall be set to:

79549 S_IRWXO | S_IRWXG | S_IRWXU

79550 and the bits specified by the *umask* of the process shall be cleared.

79551 OPTIONS

79552 The *c99* utility shall conform to XBD [Section 12.2](#) (on page 215), except that:

- 79553 • Options can be interspersed with operands.
- 79554 • The order of specifying the *-I*, *-L*, and *-l* options, and the order of specifying *-l* options
 79555 with respect to *pathname* operands is significant.
- 79556 • Conforming applications shall specify each option separately; that is, grouping option
 79557 letters (for example, *-cO*) need not be recognized by all implementations.

79558 The following options shall be supported:

79559 *-c* Suppress the link-edit phase of the compilation, and do not remove any object files
 79560 that are produced.

79561 *-D name[=value]*

79562 Define *name* as if by a C-language **#define** directive. If no *=value* is given, a value of
 79563 1 shall be used. The *-D* option has lower precedence than the *-U* option. That is, if
 79564 *name* is used in both a *-U* and a *-D* option, *name* shall be undefined regardless of
 79565 the order of the options. Additional implementation-defined *names* may be
 79566 provided by the compiler. Implementations shall support at least 2 048 bytes of *-D*
 79567 definitions and 256 *names*.

79568	-E	Copy C-language source files to standard output, expanding all preprocessor directives; no compilation shall be performed. If any operand is not a text file, the effects are unspecified.
79569		
79570		
79571	-g	Produce symbolic information in the object or executable files; the nature of this information is unspecified, and may be modified by implementation-defined interactions with other options.
79572		
79573		
79574	-I <i>directory</i>	Change the algorithm for searching for headers whose names are not absolute pathnames to look in the directory named by the <i>directory</i> pathname before looking in the usual places. Thus, headers whose names are enclosed in double-quotes (" ") shall be searched for first in the directory of the file with the #include line, then in directories named in -I options, and last in the usual places. For headers whose names are enclosed in angle brackets ("<>"), the header shall be searched for only in directories named in -I options and then in the usual places. Directories named in -I options shall be searched in the order specified. Implementations shall support at least ten instances of this option in a single c99 command invocation.
79575		
79576		
79577		
79578		
79579		
79580		
79581		
79582		
79583	-L <i>directory</i>	Change the algorithm of searching for the libraries named in the -l objects to look in the directory named by the <i>directory</i> pathname before looking in the usual places. Directories named in -L options shall be searched in the order specified. Implementations shall support at least ten instances of this option in a single c99 command invocation. If a directory specified by a -L option contains files with names starting with any of the strings "libc.", "libl.", "libpthread.", "libm.", "librt.", "libtrace.", "libxnet.", or "liby.", the results are unspecified.
79584		
79585		
79586		
79587		
79588		
79589		
79590		
79591	-l <i>library</i>	Search the library named liblibrary.a . A library shall be searched when its name is encountered, so the placement of a -l option is significant. Several standard libraries can be specified in this manner, as described in the EXTENDED DESCRIPTION section. Implementations may recognize implementation-defined suffixes other than a as denoting libraries.
79592		
79593		
79594		
79595		
79596	-O <i>optlevel</i>	Specify the level of code optimization. If the <i>optlevel</i> option-argument is the digit '0', all special code optimizations shall be disabled. If it is the digit '1', the nature of the optimization is unspecified. If the -O option is omitted, the nature of the system's default optimization is unspecified. It is unspecified whether code generated in the presence of the -O 0 option is the same as that generated when -O is omitted. Other <i>optlevel</i> values may be supported.
79597		
79598		
79599		
79600		
79601		
79602	-o <i>outfile</i>	Use the pathname <i>outfile</i> , instead of the default a.out , for the executable file produced. If the -o option is present with -c or -E, the result is unspecified.
79603		
79604	-s	Produce object or executable files, or both, from which symbolic and other information not required for proper execution using the <i>exec</i> family defined in the System Interfaces volume of POSIX.1-2008 has been removed (stripped). If both -g and -s options are present, the action taken is unspecified.
79605		
79606		
79607		
79608	-U <i>name</i>	Remove any initial definition of <i>name</i> .
79609		Multiple instances of the -D, -I, -L, -l, and -U options can be specified.
79610	OPERANDS	
79611	The application shall ensure that at least one <i>pathname</i> operand is specified. The following forms	
79612	for <i>pathname</i> operands shall be supported:	

79613	<i>file.c</i>	A C-language source file to be compiled and optionally linked. The application shall ensure that the operand is of this form if the <code>-c</code> option is used.
79614		
79615	<i>file.a</i>	A library of object files typically produced by the <i>ar</i> utility, and passed directly to the link editor. Implementations may recognize implementation-defined suffixes other than <i>.a</i> as denoting object file libraries.
79616		
79617		
79618	<i>file.o</i>	An object file produced by <i>c99 -c</i> and passed directly to the link editor. Implementations may recognize implementation-defined suffixes other than <i>.o</i> as denoting object files.
79619		
79620		
79621		The processing of other files is implementation-defined.
79622	STDIN	
79623		Not used.
79624	INPUT FILES	
79625		The input file shall be one of the following: a text file containing a C-language source program, an object file in the format produced by <i>c99 -c</i> , or a library of object files, in the format produced by archiving zero or more object files, using <i>ar</i> . Implementations may supply additional utilities that produce files in these formats. Additional input file formats are implementation-defined.
79626		
79627		
79628		
79629	ENVIRONMENT VARIABLES	
79630		The following environment variables shall affect the execution of <i>c99</i> :
79631	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
79632		
79633		
79634	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
79635		
79636	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).
79637		
79638		
79639	<i>LC_MESSAGES</i>	
79640		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
79641		
79642	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
79643	<i>TMPDIR</i>	Provide a pathname that should override the default directory for temporary files, if any. On XSI-conforming systems, provide a pathname that shall override the default directory for temporary files, if any.
79644	XSI	
79645		
79646	ASYNCHRONOUS EVENTS	
79647		Default.
79648	STDOUT	
79649		If more than one <i>pathname</i> operand ending in <i>.c</i> (or possibly other unspecified suffixes) is given, for each such file:
79650		
79651		<code>"%s:\n", <pathname></code>
79652		may be written. These messages, if written, shall precede the processing of each input file; they shall not be written to the standard output if they are written to the standard error, as described in the STDERR section.
79653		
79654		
79655		If the <code>-E</code> option is specified, the standard output shall be a text file that represents the results of

the preprocessing stage of the language; it may contain extra information appropriate for subsequent compilation passes.

STDERR

The standard error shall be used only for diagnostic messages. If more than one *pathname* operand ending in *.c* (or possibly other unspecified suffixes) is given, for each such file:

"%s:\n", <pathname>

may be written to allow identification of the diagnostic and warning messages with the appropriate input file. These messages, if written, shall precede the processing of each input file; they shall not be written to the standard error if they are written to the standard output, as described in the STDOUT section.

This utility may produce warning messages about certain conditions that do not warrant returning an error (non-zero) exit value.

OUTPUT FILES

Object files or executable files or both are produced in unspecified formats. If the pathname of an object file or executable file to be created by *c99* resolves to an existing directory entry for a file that is not a regular file, it is unspecified whether *c99* shall attempt to create the file or shall issue a diagnostic and exit with a non-zero exit status.

EXTENDED DESCRIPTION

Standard Libraries

The *c99* utility shall recognize the following *-l* options for standard libraries:

-l c This option shall make available all interfaces referenced in the System Interfaces volume of POSIX.1-2008, with the possible exception of those interfaces listed as residing in <aio.h>, <arpa/inet.h>, <complex.h>, <fenv.h>, <math.h>, <mqueue.h>, <netdb.h>, <net/if.h>, <netinet/in.h>, <pthread.h>, <sched.h>, <semaphore.h>, <spawn.h>, <sys/socket.h>, *pthread_kill()*, and *pthread_sigmask()* in <signal.h>, <trace.h>, interfaces marked as optional in <sys/mman.h>, interfaces marked as ADV (Advisory Information) in <fcntl.h>, and interfaces beginning with the prefix *clock_* or *time_* in <time.h>. This option shall not be required to be present to cause a search of this library.

-l l This option shall make available all interfaces required by the C-language output of *lex* that are not made available through the *-l c* option.

-l pthread This option shall make available all interfaces referenced in <pthread.h> and *pthread_kill()* and *pthread_sigmask()* referenced in <signal.h>. An implementation may search this library in the absence of this option.

-l m This option shall make available all interfaces referenced in <math.h>, <complex.h>, and <fenv.h>. An implementation may search this library in the absence of this option.

-l rt This option shall make available all interfaces referenced in <aio.h>, <mqueue.h>, <sched.h>, <semaphore.h>, and <spawn.h>, interfaces marked as optional in <sys/mman.h>, interfaces marked as ADV (Advisory Information) in <fcntl.h>, and interfaces beginning with the prefix *clock_* and *time_* in <time.h>. An implementation may search this library in the absence of this option.

- 79698

OB

-l trace

This option shall make available all interfaces referenced in `<trace.h>`. An implementation may search this library in the absence of this option.
- 79699
- 79700

-l xnet

This option shall make available all interfaces referenced in `<arpa/inet.h>`, `<netdb.h>`, `<net/if.h>`, `<netinet/in.h>`, and `<sys/socket.h>`. An implementation may search this library in the absence of this option.
- 79701
- 79702
- 79703

-l y

This option shall make available all interfaces required by the C-language output of *yacc* that are not made available through the `-l c` option.
- 79704
- 79705

In the absence of options that inhibit invocation of the link editor, such as `-c` or `-E`, the *c99* utility shall cause the equivalent of a `-l c` option to be passed to the link editor after the last *pathname* operand or `-l` option, causing it to be searched after all other object files and libraries are loaded.
- 79706
- 79707
- 79708

OB

It is unspecified whether the libraries `libc.a`, `libl.a`, `libm.a`, `libpthread.a`, `librt.a`, `libtrace.a`, `libxnet.a`, or `liby.a` exist as regular files. The implementation may accept as `-l` option-arguments names of objects that do not exist as regular files.
- 79709
- 79710

79711 External Symbols

- 79712

The C compiler and link editor shall support the significance of external symbols up to a length of at least 31 bytes; the action taken upon encountering symbols exceeding the implementation-defined maximum symbol length is unspecified.
- 79713
- 79714
- 79715

The compiler and link editor shall support a minimum of 511 external symbols per source or object file, and a minimum of 4095 external symbols in total. A diagnostic message shall be written to the standard output if the implementation-defined limit is exceeded; other actions are unspecified.
- 79716
- 79717
- 79718

79719 Programming Environments

- 79720

All implementations shall support one of the following programming environments as a default. Implementations may support more than one of the following programming environments. Applications can use *sysconf()* or *getconf* to determine which programming environments are supported.
- 79721
- 79722
- 79723

79724 Table 4-4 Programming Environments: Type Sizes

79725

79726

79727

79728

79729

79730

Programming Environment <i>getconf</i> Name	Bits in int	Bits in long	Bits in pointer	Bits in off_t
_POSIX_V7_ILP32_OFF32	32	32	32	32
_POSIX_V7_ILP32_OFFBIG	32	32	32	≥64
_POSIX_V7_LP64_OFF64	32	64	64	64
_POSIX_V7_LP64_OFFBIG	≥32	≥64	≥64	≥64

- 79731

All implementations shall support one or more environments where the widths of the following types are no greater than the width of type `long`:
- 79732

79733

79734

79735

79736

79737

<code>blksize_t</code>	<code>ptrdiff_t</code>	<code>tcflag_t</code>
<code>cc_t</code>	<code>size_t</code>	<code>wchar_t</code>
<code>mode_t</code>	<code>speed_t</code>	<code>wint_t</code>
<code>nfds_t</code>	<code>ssize_t</code>	
<code>pid_t</code>	<code>suseconds_t</code>	

- 79738

The executable files created when these environments are selected shall be in a proper format for

execution by the *exec* family of functions. Each environment may be one of the ones in Table 4-4 (on page 2492), or it may be another environment. The names for the environments that meet this requirement shall be output by a *getconf* command using the `POSIX_V7_WIDTH_RESTRICTED_ENVS` argument, as a <newline>-separated list of names suitable for use with the *getconf* `-v` option. If more than one environment meets the requirement, the names of all such environments shall be output on separate lines. Any of these names can then be used in a subsequent *getconf* command to obtain the flags specific to that environment with the following suffixes added as appropriate:

`_CFLAGS` To get the C compiler flags.

`_LDFLAGS` To get the linker/loader flags.

`_LIBS` To get the libraries.

This requirement may be removed in a future version.

When this utility processes a file containing a function called *main()*, it shall be defined with a return type equivalent to `int`. Using return from the initial call to *main()* shall be equivalent (other than with respect to language scope issues) to calling *exit()* with the returned value. Reaching the end of the initial call to *main()* shall be equivalent to calling *exit(0)*. The implementation shall not declare a prototype for this function.

Implementations provide configuration strings for C compiler flags, linker/loader flags, and libraries for each supported environment. When an application needs to use a specific programming environment rather than the implementation default programming environment while compiling, the application shall first verify that the implementation supports the desired environment. If the desired programming environment is supported, the application shall then invoke *c99* with the appropriate C compiler flags as the first options for the compile, the appropriate linker/loader flags after any other options except `-I` but before any operands or `-l` options, and the appropriate libraries at the end of the operands and `-l` options.

Conforming applications shall not attempt to link together object files compiled for different programming models. Applications shall also be aware that binary data placed in shared memory or in files might not be recognized by applications built for other programming models.

Table 4-5 Programming Environments: *c99* Arguments

Programming Environment <i>getconf</i> Name	Use	<i>c99</i> Arguments <i>getconf</i> Name
<code>_POSIX_V7_ILP32_OFF32</code>	C Compiler Flags Linker/Loader Flags Libraries	<code>POSIX_V7_ILP32_OFF32_CFLAGS</code> <code>POSIX_V7_ILP32_OFF32_LDFLAGS</code> <code>POSIX_V7_ILP32_OFF32_LIBS</code>
<code>_POSIX_V7_ILP32_OFFBIG</code>	C Compiler Flags Linker/Loader Flags Libraries	<code>POSIX_V7_ILP32_OFFBIG_CFLAGS</code> <code>POSIX_V7_ILP32_OFFBIG_LDFLAGS</code> <code>POSIX_V7_ILP32_OFFBIG_LIBS</code>
<code>_POSIX_V7_LP64_OFF64</code>	C Compiler Flags Linker/Loader Flags Libraries	<code>POSIX_V7_LP64_OFF64_CFLAGS</code> <code>POSIX_V7_LP64_OFF64_LDFLAGS</code> <code>POSIX_V7_LP64_OFF64_LIBS</code>
<code>_POSIX_V7_LPBIG_OFFBIG</code>	C Compiler Flags Linker/Loader Flags Libraries	<code>POSIX_V7_LPBIG_OFFBIG_CFLAGS</code> <code>POSIX_V7_LPBIG_OFFBIG_LDFLAGS</code> <code>POSIX_V7_LPBIG_OFFBIG_LIBS</code>

In addition to the type size programming environments above, all implementations also support

79783 a multi-threaded programming environment that is orthogonal to all of the programming
79784 environments listed above. The *getconf* utility can be used to get flags for the threaded
79785 programming environment, as indicated in Table 4-6.

79786 **Table 4-6** Threaded Programming Environment: *c99* Arguments

79787	Programming Environment		c99 Arguments
79788	<i>getconf</i> Name	Use	<i>getconf</i> Name
79789	_POSIX_THREADS	C Compiler Flags	POSIX_V7_THREADS_CFLAGS
79790		Linker/Loader Flags	POSIX_V7_THREADS_LDFLAGS

79791 These programming environment flags may be used in conjunction with any of the type size
79792 programming environments supported by the implementation.

79793 **EXIT STATUS**

79794 The following exit values shall be returned:

- 79795 0 Successful compilation or link edit.
- 79796 >0 An error occurred.

79797 **CONSEQUENCES OF ERRORS**

79798 When *c99* encounters a compilation error that causes an object file not to be created, it shall write
79799 a diagnostic to standard error and continue to compile other source code operands, but it shall
79800 not perform the link phase and return a non-zero exit status. If the link edit is unsuccessful, a
79801 diagnostic message shall be written to standard error and *c99* exits with a non-zero status. A
79802 conforming application shall rely on the exit status of *c99*, rather than on the existence or mode
79803 of the executable file.

79804 **APPLICATION USAGE**

79805 Since the *c99* utility usually creates files in the current directory during the compilation process,
79806 it is typically necessary to run the *c99* utility in a directory in which a file can be created.

79807 On systems providing POSIX Conformance (see XBD Chapter 2, on page 15), *c99* is required
79808 only with the C-Language Development option; XSI-conformant systems always provide *c99*.

79809 Some historical implementations have created *.o* files when *-c* is not specified and more than
79810 one source file is given. Since this area is left unspecified, the application cannot rely on *.o* files
79811 being created, but it also must be prepared for any related *.o* files that already exist being deleted
79812 at the completion of the link edit.

79813 There is the possible implication that if a user supplies versions of the standard functions (before
79814 they would be encountered by an implicit *-I m* or explicit *-I m*), that those versions would be
79815 used in place of the standard versions. There are various reasons this might not be true
79816 (functions defined as macros, manipulations for clean name space, and so on), so the existence of
79817 files named in the same manner as the standard libraries within the *-L* directories is explicitly
79818 stated to produce unspecified behavior.

79819 All of the functions specified in the System Interfaces volume of POSIX.1-2008 may be made
79820 visible by implementations when the Standard C Library is searched. Conforming applications
79821 must explicitly request searching the other standard libraries when functions made visible by
79822 those libraries are used.

79823 In the ISO C standard the mapping from physical source characters to the C source character set
79824 is implementation-defined. Implementations may strip white-space characters before the
79825 terminating <newline> of a (physical) line as part of this mapping and, as a consequence of this,

one or more white-space characters (and no other characters) between a <backslash> character and the <newline> character that terminates the line produces implementation-defined results. Portable applications should not use such constructs.

Some c99 compilers not conforming to POSIX.1-2008 do not support trigraphs by default.

EXAMPLES

1. The following usage example compiles **foo.c** and creates the executable file **foo**:

```
c99 -o foo foo.c
```

The following usage example compiles **foo.c** and creates the object file **foo.o**:

```
c99 -c foo.c
```

The following usage example compiles **foo.c** and creates the executable file **a.out**:

```
c99 foo.c
```

The following usage example compiles **foo.c**, links it with **bar.o**, and creates the executable file **a.out**. It may also create and leave **foo.o**:

```
c99 foo.c bar.o
```

2. The following example shows how an application using threads interfaces can test for support of and use a programming environment supporting 32-bit **int**, **long**, and **pointer** types and an **off_t** type using at least 64 bits:

```
offbig_env=$(getconf _POSIX_V7_ILP32_OFFBIG)
if [ $offbig_env != "-1" ] && [ $offbig_env != "undefined" ]
then
    c99 $(getconf POSIX_V7_ILP32_OFFBIG_CFLAGS) \
        $(getconf POSIX_V7_THREADS_CFLAGS) -D_XOPEN_SOURCE=700 \
        $(getconf POSIX_V7_ILP32_OFFBIG_LDFLAGS) \
        $(getconf POSIX_V7_THREADS_LDFLAGS) foo.c -o foo \
        $(getconf POSIX_V7_ILP32_OFFBIG_LIBS) \
        -l pthread
else
    echo ILP32_OFFBIG programming environment not supported
    exit 1
fi
```

3. The following examples clarify the use and interactions of **-L** and **-l** options.

Consider the case in which module **a.c** calls function **f()** in library **libQ.a**, and module **b.c** calls function **g()** in library **libp.a**. Assume that both libraries reside in **/a/b/c**. The command line to compile and link in the desired way is:

```
c99 -L /a/b/c main.o a.c -l Q b.c -l p
```

In this case the **-L** option need only precede the first **-l** option, since both **libQ.a** and **libp.a** reside in the same directory.

Multiple **-L** options can be used when library name collisions occur. Building on the previous example, suppose that the user wants to use a new **libp.a**, in **/a/a/a**, but still wants **f()** from **/a/b/c/libQ.a**:

```
c99 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p
```

In this example, the linker searches the **-L** options in the order specified, and finds

/a/a/a/libp.a before */a/b/c/libp.a* when resolving references for **b.c**. The order of the **-l** options is still important, however.

4. The following example shows how an application can use a programming environment where the widths of the following types:

blksize_t, cc_t, mode_t, nfds_t, pid_t, ptrdiff_t, size_t, speed_t, ssize_t, suseconds_t, tcflag_t, wchar_t, wint_t

are no greater than the width of type **long**:

```
# First choose one of the listed environments ...

# ... if there are no additional constraints, the first one will do:
CENV=$(getconf POSIX_V7_WIDTH_RESTRICTED_ENVS | head -n 1)

# ... or, if an environment that supports large files is preferred,
# look for names that contain "OFF64" or "OFFBIG": (This chooses
# the last one in the list if none match.)
for CENV in $(getconf POSIX_V7_WIDTH_RESTRICTED_ENVS)
do
    case $CENV in
        *OFF64*|*OFFBIG*) break ;;
    esac
done

# The chosen environment name can now be used like this:

c99 $(getconf ${CENV}_CFLAGS) -D _POSIX_C_SOURCE=200809L \
$(getconf ${CENV}_LDFLAGS) foo.c -o foo \
$(getconf ${CENV}_LIBS)
```

RATIONALE

The **c99** utility is based on the **c89** utility originally introduced in the ISO POSIX-2:1993 standard.

Some of the changes from **c89** include the ability to intersperse options and operands (which many **c89** implementations allowed despite it not being specified), the description of **-l** as an option instead of an operand, and the modification to the contents of the Standard Libraries section to account for new headers and options; for example, **<spawn.h>** added to the description of **-l rt**, and **-l trace** added for the Tracing option.

POSIX.1-2008 specifies that the **c99** utility must be able to use regular files for ***.o** files and for **a.out** files. Implementations are free to overwrite existing files of other types when attempting to create object files and executable files, but are not required to do so. If something other than a regular file is specified and using it fails for any reason, **c99** is required to issue a diagnostic message and exit with a non-zero exit status. But for some file types, the problem may not be noticed for a long time. For example, if a FIFO named **a.out** exists in the current directory, **c99** may attempt to open **a.out** and will hang in the **open()** call until another process opens the FIFO for reading. Then **c99** may write most of the **a.out** to the FIFO and fail when it tries to seek back close to the start of the file to insert a timestamp (FIFOs are not seekable files). The **c99** utility is also allowed to issue a diagnostic immediately if it encounters an **a.out** or ***.o** file that is not a regular file. For portable use, applications should ensure that any **a.out**, **-o** option-argument, or ***.o** files corresponding to any ***.c** files do not conflict with names already in use that are not regular files or symbolic links that point to regular files.

On many systems, multi-threaded applications run in a programming environment that is

distinct from that used by single-threaded applications. This multi-threaded programming environment (in addition to needing to specify `-l pthread` at link time) may require additional flags to be set when headers are processed at compile time (`-D_REENTRANT` being common). This programming environment is orthogonal to the type size programming environments discussed above and listed in Table 4-4 (on page 2492). This version of the standard adds *getconf* utility calls to provide the C compiler flags and linker/loader flags needed to support multi-threaded applications. Note that on a system where single-threaded applications are a special case of a multi-threaded application, both of these *getconf* calls may return NULL strings; on other implementations both of these strings may be non-NULL strings.

The C standardization committee invented trigraphs (e.g., "??!" to represent '~') to address character portability problems in development environments based on national variants of the 7-bit ISO/IEC 646:1991 standard character set. However, these environments were already obsolete by the time the first ISO C standard was published, and in practice trigraphs have not been used for their intended purpose, and usually are intended to have their original meaning in K&R C. For example, in practice a C-language source string like "What??!" is usually intended to end in two <question-mark> characters and an <exclamation-mark>, not in '~'.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 1.1.1.4 (on page 2280), *ar*, *getconf*, *make*, *nm*, *strip*, *umask*

XBD Chapter 8 (on page 173), Section 12.2 (on page 215), Chapter 13 (on page 219)

XSH *exec*, *sysconf()*

CHANGE HISTORY

First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/12 is applied, correcting the EXTENDED DESCRIPTION of `-l c` and `-l m`. Previously, the text did not take into account the presence of the c99 math headers.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/13 is applied, changing the reference to the *libxnet* library to *libxnet.a*.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/5 is applied, updating the OPTIONS section, so that the names of files contained in the directory specified by the `-L` option are not assumed to end in the *.a* suffix. The set of library prefixes is also updated.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/6 is applied, removing the lead underscore from the `POSIX_V6_WIDTH_RESTRICTED_ENVS` variable in the EXTENDED DESCRIPTION and the EXAMPLES sections.

Issue 7

Austin Group Interpretation 1003.1-2001 #020 (SD5-XCU-ERN-10) is applied, adding to the OUTPUT FILES section and also adding associated RATIONALE.

Austin Group Interpretation 1003.1-2001 #095 is applied, clarifying the `-l library` operand.

Austin Group Interpretation 1003.1-2001 #166 is applied.

Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the handling of trailing white-space characters.

Austin Group Interpretation 1003.1-2001 #191 is applied, adding APPLICATION USAGE and RATIONALE regarding C-language trigraphs.

79957	SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not
79958	apply (options can be interspersed with operands).
79959	SD5-XCU-ERN-11 is applied, adding the <net/if.h> header to the descriptions of -l c and
79960	-l xnet .
79961	SD5-XCU-ERN-65 is applied, updating the EXAMPLES section.
79962	SD5-XCU-ERN-67 and SD5-XCU-ERN-97 are applied, updating the SYNOPSIS.
79963	SD5-XCU-ERN-133 is applied, updating the EXTENDED DESCRIPTION.
79964	The <i>getconf</i> variables for the supported programming environments are updated to be V7.
79965	The -l trace operand is marked obsolescent.
79966	The <i>c99</i> reference page is rewritten to describe -l as an option rather than an operand.

Utilities

cal

79967 **NAME**

79968 cal — print a calendar

79969 **SYNOPSIS**79970 XSI cal **[[month] year]**79971 **DESCRIPTION**

79972 The *cal* utility shall write a calendar to standard output using the Julian calendar for dates from
 79973 January 1, 1 through September 2, 1752 and the Gregorian calendar for dates from September 14,
 79974 1752 through December 31, 9999 as though the Gregorian calendar had been adopted on
 79975 September 14, 1752.

79976 **OPTIONS**

79977 None.

79978 **OPERANDS**

79979 The following operands shall be supported:

79980 *month* Specify the month to be displayed, represented as a decimal integer from 1
 79981 (January) to 12 (December). The default shall be the current month.

79982 *year* Specify the year for which the calendar is displayed, represented as a decimal
 79983 integer from 1 to 9999. The default shall be the current year.

79984 **STDIN**

79985 Not used.

79986 **INPUT FILES**

79987 None.

79988 **ENVIRONMENT VARIABLES**79989 The following environment variables shall affect the execution of *cal*:

79990 *LANG* Provide a default value for the internationalization variables that are unset or null.
 79991 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 79992 variables used to determine the values of locale categories.)

79993 *LC_ALL* If set to a non-empty string value, override the values of all the other
 79994 internationalization variables.

79995 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 79996 characters (for example, single-byte as opposed to multi-byte characters in
 79997 arguments).

79998 *LC_MESSAGES*

79999 Determine the locale that should be used to affect the format and contents of
 80000 diagnostic messages written to standard error, and informative messages written
 80001 to standard output.

80002 *LC_TIME* Determine the format and contents of the calendar.

80003 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

80004 *TZ* Determine the timezone used to calculate the value of the current month.

80005 **ASYNCHRONOUS EVENTS**

80006 Default.

cal*Utilities***80007 STDOUT**

80008 The standard output shall be used to display the calendar, in an unspecified format.

80009 STDERR

80010 The standard error shall be used only for diagnostic messages.

80011 OUTPUT FILES

80012 None.

80013 EXTENDED DESCRIPTION

80014 None.

80015 EXIT STATUS

80016 The following exit values shall be returned:

80017 0 Successful completion.

80018 >0 An error occurred.

80019 CONSEQUENCES OF ERRORS

80020 Default.

80021 APPLICATION USAGE

80022 Note that:

80023 cal 83

80024 refers to A.D. 83, not 1983.

80025 EXAMPLES

80026 None.

80027 RATIONALE

80028 None.

80029 FUTURE DIRECTIONS

80030 A future version of this standard may support locale-specific recognition of the date of adoption
80031 of the Gregorian calendar.

80032 SEE ALSO

80033 XBD [Chapter 8](#) (on page 173)

80034 CHANGE HISTORY

80035 First released in Issue 2.

80036 Issue 6

80037 The DESCRIPTION is updated to allow for traditional behavior for years before the adoption of
80038 the Gregorian calendar.

80039 Issue 7

80040 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Utilities

cat

80041 NAME

80042 cat — concatenate and print files

80043 SYNOPSIS

80044 cat [-u] [*file*...]

80045 DESCRIPTION

80046 The *cat* utility shall read files in sequence and shall write their contents to the standard output in
80047 the same sequence.

80048 OPTIONS

80049 The *cat* utility shall conform to XBD Section 12.2 (on page 215).

80050 The following option shall be supported:

80051 **-u** Write bytes from the input file to the standard output without delay as each is
80052 read.

80053 OPERANDS

80054 The following operand shall be supported:

80055 *file* A pathname of an input file. If no *file* operands are specified, the standard input
80056 shall be used. If a *file* is '-', the *cat* utility shall read from the standard input at
80057 that point in the sequence. The *cat* utility shall not close and reopen standard input
80058 when it is referenced in this way, but shall accept multiple occurrences of '-' as a
80059 *file* operand.

80060 STDIN

80061 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
80062 See the INPUT FILES section.

80063 INPUT FILES

80064 The input files can be any file type.

80065 ENVIRONMENT VARIABLES

80066 The following environment variables shall affect the execution of *cat*:

80067 **LANG** Provide a default value for the internationalization variables that are unset or null.
80068 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
80069 variables used to determine the values of locale categories.)

80070 **LC_ALL** If set to a non-empty string value, override the values of all the other
80071 internationalization variables.

80072 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
80073 characters (for example, single-byte as opposed to multi-byte characters in
80074 arguments).

80075 **LC_MESSAGES**

80076 Determine the locale that should be used to affect the format and contents of
80077 diagnostic messages written to standard error.

80078 **XSI NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

80079 ASYNCHRONOUS EVENTS

80080 Default.

80081 **STDOUT**

80082 The standard output shall contain the sequence of bytes read from the input files. Nothing else
80083 shall be written to the standard output.

80084 **STDERR**

80085 The standard error shall be used only for diagnostic messages.

80086 **OUTPUT FILES**

80087 None.

80088 **EXTENDED DESCRIPTION**

80089 None.

80090 **EXIT STATUS**

80091 The following exit values shall be returned:

80092 0 All input files were output successfully.

80093 >0 An error occurred.

80094 **CONSEQUENCES OF ERRORS**

80095 Default.

80096 **APPLICATION USAGE**

80097 The **-u** option has value in prototyping non-blocking reads from FIFOs. The intent is to support
80098 the following sequence:

```
80099 mkfifo foo
80100 cat -u foo > /dev/tty13 &
80101 cat -u > foo
```

80102 It is unspecified whether standard output is or is not buffered in the default case. This is
80103 sometimes of interest when standard output is associated with a terminal, since buffering may
80104 delay the output. The presence of the **-u** option guarantees that unbuffered I/O is available. It is
80105 implementation-defined whether the *cat* utility buffers output if the **-u** option is not specified.
80106 Traditionally, the **-u** option is implemented using the equivalent of the *setvbuf()* function
80107 defined in the System Interfaces volume of POSIX.1-2008.

80108 **EXAMPLES**

80109 The following command:

```
80110 cat myfile
```

80111 writes the contents of the file **myfile** to standard output.

80112 The following command:

```
80113 cat doc1 doc2 > doc.all
```

80114 concatenates the files **doc1** and **doc2** and writes the result to **doc.all**.

80115 Because of the shell language mechanism used to perform output redirection, a command such
80116 as this:

```
80117 cat doc doc.end > doc
```

80118 causes the original data in **doc** to be lost.

80119 The command:

```
80120 cat start - middle - end > file
```

80121 when standard input is a terminal, gets two arbitrary pieces of input from the terminal with a

single invocation of *cat*. Note, however, that if standard input is a regular file, this would be equivalent to the command:

```
cat start - middle /dev/null end > file
```

because the entire contents of the file would be consumed by *cat* the first time '-' was used as a *file* operand and an end-of-file condition would be detected immediately when '-' was referenced the second time.

RATIONALE

Historical versions of the *cat* utility include the *-e*, *-t*, and *-v*, options which permit the ends of lines, <tab> characters, and invisible characters, respectively, to be rendered visible in the output. The standard developers omitted these options because they provide too fine a degree of control over what is made visible, and similar output can be obtained using a command such as:

```
sed -n l pathname
```

The latter also has the advantage that its output is unambiguous, whereas the output of historical *cat -etv* is not.

The *-s* option was omitted because it corresponds to different functions in BSD and System V-based systems. The BSD *-s* option to squeeze blank lines can be accomplished by the shell script shown in the following example:

```
sed -n '
# Write non-empty lines.
./ {
    p
    d
}
# Write a single empty line, then look for more empty lines.
/^$/ p
# Get next line, discard the held <newline> (empty line),
# and look for more empty lines.
:Empty
/^$/ {
    N
    s/./ /
    b Empty
}
# Write the non-empty line before going back to search
# for the first in a set of empty lines.
p
'
```

The System V *-s* option to silence error messages can be accomplished by redirecting the standard error. Note that the BSD documentation for *cat* uses the term "blank line" to mean the same as the POSIX "empty line": a line consisting only of a <newline>.

The BSD *-n* option was omitted because similar functionality can be obtained from the *-n* option of the *pr* utility.

FUTURE DIRECTIONS

None.

cat*Utilities*80166 **SEE ALSO**80167 [more](#)80168 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)80169 XSH [setvbuf\(\)](#)80170 **CHANGE HISTORY**

80171 First released in Issue 2.

80172 **Issue 7**

80173 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80174 SD5-XCU-ERN-174 is applied, changing the RATIONALE.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

cd

NAME

cd — change the working directory

SYNOPSIS

cd [-L|-P] [*directory*]

cd -

DESCRIPTION

The *cd* utility shall change the working directory of the current shell execution environment (see [Section 2.12](#), on page 2331) by executing the following steps in sequence. (In the following steps, the symbol **curpath** represents an intermediate value used to simplify the description of the algorithm used by *cd*. There is no requirement that **curpath** be made visible to the application.)

1. If no *directory* operand is given and the *HOME* environment variable is empty or undefined, the default behavior is implementation-defined and no further steps shall be taken.
2. If no *directory* operand is given and the *HOME* environment variable is set to a non-empty value, the *cd* utility shall behave as if the directory named in the *HOME* environment variable was specified as the *directory* operand.
3. If the *directory* operand begins with a <slash> character, set **curpath** to the operand and proceed to step 7.
4. If the first component of the *directory* operand is dot or dot-dot, proceed to step 6.
5. Starting with the first pathname in the <colon>-separated pathnames of *CDPATH* (see the ENVIRONMENT VARIABLES section), if the pathname is non-null, test if the concatenation of that pathname, a <slash> character if that pathname did not end with a <slash> character, and the *directory* operand names a directory. If the pathname is null, test if the concatenation of dot, a <slash> character, and the operand names a directory. In either case, if the resulting string names an existing directory, set **curpath** to that string and proceed to step 7. Otherwise, repeat this step with the next pathname in *CDPATH* until all pathnames have been tested.
6. If the *-P* option is in effect, set **curpath** to the *directory* operand. Otherwise, set **curpath** to the string formed by the concatenation of the value of *PWD*, a <slash> character if the value of *PWD* did not end with a <slash> character, and the operand.
7. If the *-P* option is in effect, proceed to step 10. If **curpath** does not begin with a <slash> character, set **curpath** to the string formed by the concatenation of the value of *PWD*, a <slash> character if the value of *PWD* did not end with a <slash> character, and **curpath**.
8. The **curpath** value shall then be converted to canonical form as follows, considering each component from beginning to end, in sequence:
 - a. Dot components and any <slash> characters that separate them from the next component shall be deleted.
 - b. For each dot-dot component, if there is a preceding component and it is neither root nor dot-dot, then:
 - i. If the preceding component does not refer (in the context of pathname resolution with symbolic links followed) to a directory, then the *cd* utility shall display an appropriate error message and no further steps shall be taken.

cd*Utilities*

- 80218 ii. The preceding component, all <slash> characters separating the preceding
80219 component from dot-dot, dot-dot, and all <slash> characters separating dot-
80220 dot from the following component (if any) shall be deleted.
- 80221 c. An implementation may further simplify **curpath** by removing any trailing
80222 <slash> characters that are not also leading <slash> characters, replacing multiple
80223 non-leading consecutive <slash> characters with a single <slash>, and replacing
80224 three or more leading <slash> characters with a single <slash>. If, as a result of
80225 this canonicalization, the **curpath** variable is null, no further steps shall be taken.
- 80226 9. If **curpath** is longer than {PATH_MAX} bytes (including the terminating null) and the
80227 *directory* operand was not longer than {PATH_MAX} bytes (including the terminating
80228 null), then **curpath** shall be converted from an absolute pathname to an equivalent
80229 relative pathname if possible. This conversion shall always be considered possible if the
80230 value of *PWD*, with a trailing <slash> added if it does not already have one, is an initial
80231 substring of **curpath**. Whether or not it is considered possible under other circumstances
80232 is unspecified. Implementations may also apply this conversion if **curpath** is not longer
80233 than {PATH_MAX} bytes or the *directory* operand was longer than {PATH_MAX} bytes.
- 80234 10. The *cd* utility shall then perform actions equivalent to the *chdir()* function called with
80235 **curpath** as the *path* argument. If these actions fail for any reason, the *cd* utility shall
80236 display an appropriate error message and the remainder of this step shall not be
80237 executed. If the **-P** option is not in effect, the *PWD* environment variable shall be set to
80238 the value that **curpath** had on entry to step 9 (i.e., before conversion to a relative
80239 pathname). If the **-P** option is in effect, the *PWD* environment variable shall be set to the
80240 string that would be output by *pwd -P*. If there is insufficient permission on the new
80241 directory, or on any parent of that directory, to determine the current working directory,
80242 the value of the *PWD* environment variable is unspecified.

80243 If, during the execution of the above steps, the *PWD* environment variable is changed, the
80244 *OLDPWD* environment variable shall also be changed to the value of the old working directory
80245 (that is the current working directory immediately prior to the call to *cd*).

OPTIONS

80246 The *cd* utility shall conform to XBD [Section 12.2](#) (on page 215).

80247 The following options shall be supported by the implementation:

- 80248 **-L** Handle the operand dot-dot logically; symbolic link components shall not be
80249 resolved before dot-dot components are processed (see steps 8. and 9. in the
80250 DESCRIPTION).
- 80251 **-P** Handle the operand dot-dot physically; symbolic link components shall be
80252 resolved before dot-dot components are processed (see step 7. in the
80253 DESCRIPTION).

80254 If both **-L** and **-P** options are specified, the last of these options shall be used and all others
80255 ignored. If neither **-L** nor **-P** is specified, the operand shall be handled dot-dot logically; see the
80256 DESCRIPTION.

OPERANDS

80258 The following operands shall be supported:

- 80259 *directory* An absolute or relative pathname of the directory that shall become the new
80260 working directory. The interpretation of a relative pathname by *cd* depends on the
80261 **-L** option and the *CDPATH* and *PWD* environment variables. If *directory* is an
80262 empty string, the results are unspecified.

Utilities

cd

80264	–	When a <hyphen> is used as the operand, this shall be equivalent to the command:
80265		<code>cd "\$OLDPWD" && pwd</code>
80266		which changes to the previous working directory and then writes its name.
80267	STDIN	
80268		Not used.
80269	INPUT FILES	
80270		None.
80271	ENVIRONMENT VARIABLES	
80272		The following environment variables shall affect the execution of <i>cd</i> :
80273	<i>CDPATH</i>	A <colon>-separated list of pathnames that refer to directories. The <i>cd</i> utility shall
80274		use this list in its attempt to change the directory, as described in the
80275		DESCRIPTION. An empty string in place of a directory pathname represents the
80276		current directory. If <i>CDPATH</i> is not set, it shall be treated as if it were an empty
80277		string.
80278	<i>HOME</i>	The name of the directory, used when no <i>directory</i> operand is specified.
80279	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null.
80280		(See XBD Section 8.2 (on page 174) for the precedence of internationalization
80281		variables used to determine the values of locale categories.)
80282	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other
80283		internationalization variables.
80284	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as
80285		characters (for example, single-byte as opposed to multi-byte characters in
80286		arguments).
80287	<i>LC_MESSAGES</i>	
80288		Determine the locale that should be used to affect the format and contents of
80289		diagnostic messages written to standard error.
80290	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
80291	<i>OLDPWD</i>	A pathname of the previous working directory, used by <i>cd –</i> .
80292	<i>PWD</i>	This variable shall be set as specified in the DESCRIPTION. If an application sets
80293		or unsets the value of <i>PWD</i> , the behavior of <i>cd</i> is unspecified.
80294	ASYNCHRONOUS EVENTS	
80295		Default.
80296	STDOUT	
80297		If a non-empty directory name from <i>CDPATH</i> is used, or if <i>cd –</i> is used, an absolute pathname of
80298		the new working directory shall be written to the standard output as follows:
80299		<code>"%s\n", <new directory></code>
80300		Otherwise, there shall be no output.
80301	STDERR	
80302		The standard error shall be used only for diagnostic messages.

cd*Utilities*80303 **OUTPUT FILES**

80304 None.

80305 **EXTENDED DESCRIPTION**

80306 None.

80307 **EXIT STATUS**

80308 The following exit values shall be returned:

80309 0 The directory was successfully changed.

80310 >0 An error occurred.

80311 **CONSEQUENCES OF ERRORS**

80312 The working directory shall remain unchanged.

80313 **APPLICATION USAGE**

80314 Since *cd* affects the current shell execution environment, it is always provided as a shell regular
 80315 built-in. If it is called in a subshell or separate utility execution environment, such as one of the
 80316 following:

```
80317 (cd /tmp)
80318 nohup cd
80319 find . -exec cd {} \;
```

80320 it does not affect the working directory of the caller's environment.

80321 The user must have execute (search) permission in *directory* in order to change to it.80322 **EXAMPLES**

80323 None.

80324 **RATIONALE**

80325 The use of the *CDPATH* was introduced in the System V shell. Its use is analogous to the use of
 80326 the *PATH* variable in the shell. The BSD C shell used a shell parameter *cdpath* for this purpose.

80327 A common extension when *HOME* is undefined is to get the login directory from the user
 80328 database for the invoking user. This does not occur on System V implementations.

80329 Some historical shells, such as the KornShell, took special actions when the directory name
 80330 contained a dot-dot component, selecting the logical parent of the directory, rather than the
 80331 actual parent directory; that is, it moved up one level toward the '/' in the pathname,
 80332 remembering what the user typed, rather than performing the equivalent of:

80333 `chdir("../");`

80334 In such a shell, the following commands would not necessarily produce equivalent output for all
 80335 directories:

80336 `cd .. && ls           ls ..`

80337 This behavior is now the default. It is not consistent with the definition of dot-dot in most
 80338 historical practice; that is, while this behavior has been optionally available in the KornShell,
 80339 other shells have historically not supported this functionality. The logical pathname is stored in
 80340 the *PWD* environment variable when the *cd* utility completes and this value is used to construct
 80341 the next directory name if *cd* is invoked with the *-L* option.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.12 (on page 2331), *pwd*

XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

XSH *chdir()*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *cd* – operand, *PWD*, and *OLDPWD* are added.

The *–L* and *–P* options are added to align with the IEEE P1003.2b draft standard. This also includes the introduction of a new description to include the effect of these options.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/14 is applied, changing the SYNOPSIS to make it clear that the *–L* and *–P* options are mutually-exclusive.

Issue 7

Austin Group Interpretation 1003.1-2001 #037 is applied.

Austin Group Interpretation 1003.1-2001 #199 is applied, clarifying how the *od* utility handles concatenation of two pathnames when the first pathname ends in a <slash> character.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Step 7 of the processing performed by *cd* is revised to refer to **curpath** instead of “the operand”.

Changes to the *pwd* utility and *PWD* environment variable have been made to match the changes to the *getcwd()* function made for Austin Group Interpretation 1003.1-2001 #140.

cflow

Utilities

NAMEcflow — generate a C-language flowgraph (**DEVELOPMENT**)**SYNOPSIS**

```

cflow [-r] [-d num] [-D name[=def]]... [-i incl] [-I dir]...
      [-U dir]... file...

```

DESCRIPTION

The *cflow* utility shall analyze a collection of object files or assembler, C-language, *lex*, or *yacc* source files, and attempt to build a graph, written to standard output, charting the external references.

OPTIONS

The *cflow* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the order of the **-D**, **-I**, and **-U** options (which are identical to their interpretation by *c99*) is significant.

The following options shall be supported:

- d num** Indicate the depth at which the flowgraph is cut off. The application shall ensure that the argument *num* is a decimal integer. By default this is a very large number (typically greater than 32 000). Attempts to set the cut-off depth to a non-positive integer shall be ignored.
- i incl** Increase the number of included symbols. The *incl* option-argument is one of the following characters:
 - x* Include external and static data symbols. The default shall be to include only functions in the flowgraph.
 - _* (Underscore) Include names that begin with an <underscore>. The default shall be to exclude these functions (and data if **-i x** is used).
- r** Reverse the caller: callee relationship, producing an inverted listing showing the callers of each function. The listing shall also be sorted in lexicographical order by callee.

OPERANDS

The following operand is supported:

- file* The pathname of a file for which a graph is to be generated. Filenames suffixed by **.l** shall be taken to be *lex* input, **.y** as *yacc* input, **.c** as *c99* input, and **.i** as the output of *c99* **-E**. Such files shall be processed as appropriate, determined by their suffix.
- Files suffixed by **.s** (conventionally assembler source) may have more limited information extracted from them.

STDIN

Not used.

INPUT FILES

The input files shall be object files or assembler, C-language, *lex*, or *yacc* source files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *cflow*:

- LANG** Provide a default value for the internationalization variables that are unset or null. (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

Utilities

cflow

80409	LC_ALL	If set to a non-empty string value, override the values of all the other internationalization variables.
80410		
80411	LC_COLLATE	
80412		Determine the locale for the ordering of the output when the -r option is used.
80413	LC_CTYPE	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).
80414		
80415		
80416	LC_MESSAGES	
80417		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
80418		
80419	NLSPATH	Determine the location of message catalogs for the processing of LC_MESSAGES .
80420	ASYNCHRONOUS EVENTS	
80421		Default.
80422	STDOUT	
80423		The flowgraph written to standard output shall be formatted as follows:
80424		"%d %s:%s\n", <reference number>, <global>, <definition>
80425		Each line of output begins with a reference (that is, line) number, followed by indentation of at least one column position per level. This is followed by the name of the global, a <colon>, and its definition. Normally globals are only functions not defined as an external or beginning with an <underscore>; see the OPTIONS section for the -i inclusion option. For information extracted from C-language source, the definition consists of an abstract type declaration (for example, char *) and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the filename and location counter under which the symbol appeared (for example, <i>text</i>).
80426		
80427		
80428		
80429		
80430		
80431		
80432		
80433		Once a definition of a name has been written, subsequent references to that name contain only the reference number of the line where the definition can be found. For undefined references, only "<>" shall be written.
80434		
80435		
80436	STDERR	
80437		The standard error shall be used only for diagnostic messages.
80438	OUTPUT FILES	
80439		None.
80440	EXTENDED DESCRIPTION	
80441		None.
80442	EXIT STATUS	
80443		The following exit values shall be returned:
80444	0	Successful completion.
80445	>0	An error occurred.
80446	CONSEQUENCES OF ERRORS	
80447		Default.

APPLICATION USAGE

Files produced by *lex* and *yacc* cause the reordering of line number declarations, and this can confuse *cflow*. To obtain proper results, the input of *yacc* or *lex* must be directed to *cflow*.

EXAMPLES

Given the following in **file.c**:

```
int i;
int f();
int g();
int h();
int
main()
{
    f();
    g();
    f();
}
int
f()
{
    i = h();
}
```

The command:

```
cflow -i x file.c
```

produces the output:

```
1 main: int(), <file.c 6>
2   f: int(), <file.c 13>
3     h: <>
4     i: int, <file.c 1>
5   g: <>
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

c99, *lex*, *yacc*

XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term “must” for application requirements.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

NAME

chgrp — change the file group ownership

SYNOPSIS

chgrp [-h] *group file...*

chgrp -R [-H|-L|-P] *group file...*

DESCRIPTION

The *chgrp* utility shall set the group ID of the file named by each *file* operand to the group ID specified by the *group* operand.

For each *file* operand, or, if the *-R* option is used, each file encountered while walking the directory trees specified by the *file* operands, the *chgrp* utility shall perform actions equivalent to the *chown()* function defined in the System Interfaces volume of POSIX.1-2008, called with the following arguments:

- The *file* operand shall be used as the *path* argument.
- The user ID of the file shall be used as the *owner* argument.
- The specified group ID shall be used as the *group* argument.

Unless *chgrp* is invoked by a process with appropriate privileges, the set-user-ID and set-group-ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-group-ID bits of other file types may be cleared.

OPTIONS

The *chgrp* utility shall conform to XBD [Section 12.2](#) (on page 215).

The following options shall be supported by the implementation:

- h** If the system supports group IDs for symbolic links, for each *file* operand that names a file of type symbolic link, *chgrp* shall attempt to set the group ID of the symbolic link instead of the file referenced by the symbolic link. If the system does not support group IDs for symbolic links, for each *file* operand that names a file of type symbolic link, *chgrp* shall do nothing more with the current file and shall go on to any remaining files.
- H** If the *-R* option is specified and a symbolic link referencing a file of type directory is specified on the command line, *chgrp* shall change the group of the directory referenced by the symbolic link and all files in the file hierarchy below it.
- L** If the *-R* option is specified and a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, *chgrp* shall change the group of the directory referenced by the symbolic link and all files in the file hierarchy below it.
- P** If the *-R* option is specified and a symbolic link is specified on the command line or encountered during the traversal of a file hierarchy, *chgrp* shall change the group ID of the symbolic link if the system supports this operation. The *chgrp* utility shall not follow the symbolic link to any other part of the file hierarchy.
- R** Recursively change file group IDs. For each *file* operand that names a directory, *chgrp* shall change the group of the directory and all files in the file hierarchy below it. Unless a *-H*, *-L*, or *-P* option is specified, it is unspecified which of these options will be used as the default.

Specifying more than one of the mutually-exclusive options *-H*, *-L*, and *-P* shall not be considered an error. The last option specified shall determine the behavior of the utility.

chgrp

Utilities

80534 **OPERANDS**

80535 The following operands shall be supported:

80536 *group* A group name from the group database or a numeric group ID. Either specifies a
 80537 group ID to be given to each file named by one of the *file* operands. If a numeric
 80538 *group* operand exists in the group database as a group name, the group ID number
 80539 associated with that group name is used as the group ID.

80540 *file* A pathname of a file whose group ID is to be modified.

80541 **STDIN**

80542 Not used.

80543 **INPUT FILES**

80544 None.

80545 **ENVIRONMENT VARIABLES**80546 The following environment variables shall affect the execution of *chgrp*:

80547 *LANG* Provide a default value for the internationalization variables that are unset or null.
 80548 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 80549 variables used to determine the values of locale categories.)

80550 *LC_ALL* If set to a non-empty string value, override the values of all the other
 80551 internationalization variables.

80552 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 80553 characters (for example, single-byte as opposed to multi-byte characters in
 80554 arguments).

80555 *LC_MESSAGES*

80556 Determine the locale that should be used to affect the format and contents of
 80557 diagnostic messages written to standard error.

80558 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

80559 **ASYNCHRONOUS EVENTS**

80560 Default.

80561 **STDOUT**

80562 Not used.

80563 **STDERR**

80564 The standard error shall be used only for diagnostic messages.

80565 **OUTPUT FILES**

80566 None.

80567 **EXTENDED DESCRIPTION**

80568 None.

80569 **EXIT STATUS**

80570 The following exit values shall be returned:

80571 0 The utility executed successfully and all requested changes were made.

80572 >0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Only the owner of a file or the user with appropriate privileges may change the owner or group of a file.

Some implementations restrict the use of *chgrp* to a user with appropriate privileges when the *group* specified is not the effective group ID or one of the supplementary group IDs of the calling process.

EXAMPLES

None.

RATIONALE

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. The standard developers chose to mask these by specifying only 0 and >0 as exit values.

The functionality of *chgrp* is described substantially through references to *chown()*. In this way, there is no duplication of effort required for describing the interactions of permissions, multiple groups, and so on.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod, *chown*

XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

XSH *chown()*

CHANGE HISTORY

First released in Issue 2.

Issue 6

New options **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links.

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section to "Default".

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/15 is applied, changing the SYNOPSIS to make it clear that **-h** and **-R** are optional.

Issue 7

SD5-XCU-ERN-8 is applied, removing the **-R** from the first line of the SYNOPSIS.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

chmod

Utilities

80609 **NAME**80610 `chmod` — change the file modes80611 **SYNOPSIS**80612 `chmod [-R] mode file...`80613 **DESCRIPTION**80614 The *chmod* utility shall change any or all of the file mode bits of the file named by each *file*
80615 operand in the way specified by the *mode* operand.80616 It is implementation-defined whether and how the *chmod* utility affects any alternate or
80617 additional file access control mechanism (see XBD [Section 4.4](#), on page 108) being used for the
80618 specified file.80619 Only a process whose effective user ID matches the user ID of the file, or a process with
80620 appropriate privileges, shall be permitted to change the file mode bits of a file.80621 Upon successfully changing the file mode bits of a file, the *chmod* utility shall mark for update
80622 the last file status change timestamp of the file.80623 **OPTIONS**80624 The *chmod* utility shall conform to XBD [Section 12.2](#) (on page 215).

80625 The following option shall be supported:

80626 **-R** Recursively change file mode bits. For each *file* operand that names a directory,
80627 *chmod* shall change the file mode bits of the directory and all files in the file
80628 hierarchy below it.80629 **OPERANDS**

80630 The following operands shall be supported:

80631 *mode* Represents the change to be made to the file mode bits of each file named by one of
80632 the *file* operands; see the EXTENDED DESCRIPTION section.80633 *file* A pathname of a file whose file mode bits shall be modified.80634 **STDIN**

80635 Not used.

80636 **INPUT FILES**

80637 None.

80638 **ENVIRONMENT VARIABLES**80639 The following environment variables shall affect the execution of *chmod*:80640 *LANG* Provide a default value for the internationalization variables that are unset or null.
80641 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
80642 variables used to determine the values of locale categories.)80643 *LC_ALL* If set to a non-empty string value, override the values of all the other
80644 internationalization variables.80645 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
80646 characters (for example, single-byte as opposed to multi-byte characters in
80647 arguments).80648 *LC_MESSAGES*80649 Determine the locale that should be used to affect the format and contents of
80650 diagnostic messages written to standard error.

Utilities

chmod

80651 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

80652 ASYNCHRONOUS EVENTS

80653 Default.

80654 STDOUT

80655 Not used.

80656 STDERR

80657 The standard error shall be used only for diagnostic messages.

80658 OUTPUT FILES

80659 None.

80660 EXTENDED DESCRIPTION

80661 The *mode* operand shall be either a *symbolic_mode* expression or a non-negative octal integer. The
80662 *symbolic_mode* form is described by the grammar later in this section.

80663 Each **clause** shall specify an operation to be performed on the current file mode bits of each *file*.
80664 The operations shall be performed on each *file* in the order in which the **clauses** are specified.

80665 The **who** symbols **u**, **g**, and **o** shall specify the *user*, *group*, and *other* parts of the file mode bits,
80666 respectively. A **who** consisting of the symbol **a** shall be equivalent to **ugo**.

80667 The **perm** symbols **r**, **w**, and **x** represent the *read*, *write*, and *execute/search* portions of file mode
80668 bits, respectively. The **perm** symbol **s** shall represent the *set-user-ID-on-execution* (when **who**
80669 contains or implies **u**) and *set-group-ID-on-execution* (when **who** contains or implies **g**) bits.

80670 The **perm** symbol **X** shall represent the *execute/search* portion of the file mode bits if the file is a
80671 directory or if the current (unmodified) file mode bits have at least one of the execute bits
80672 (**S_IXUSR**, **S_IXGRP**, or **S_IXOTH**) set. It shall be ignored if the file is not a directory and none of
80673 the execute bits are set in the current file mode bits.

80674 The **permcop** symbols **u**, **g**, and **o** shall represent the current permissions associated with the
80675 user, group, and other parts of the file mode bits, respectively. For the remainder of this section,
80676 **perm** refers to the non-terminals **perm** and **permcop** in the grammar.

80677 If multiple **actionlists** are grouped with a single **wholist** in the grammar, each **actionlist** shall be
80678 applied in the order specified with that **wholist**. The *op* symbols shall represent the operation
80679 performed, as follows:

80680 + If **perm** is not specified, the '+' operation shall not change the file mode bits.

80681 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
80682 other permissions, except for those with corresponding bits in the file mode creation mask
80683 of the invoking process, shall be set.

80684 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

80685 – If **perm** is not specified, the '-' operation shall not change the file mode bits.

80686 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
80687 other permissions, except for those with corresponding bits in the file mode creation mask
80688 of the invoking process, shall be cleared.

80689 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be
80690 cleared.

80691 = Clear the file mode bits specified by the **who** value, or, if no **who** value is specified, all of the
80692 file mode bits specified in this volume of POSIX.1-2008.

chmod*Utilities*

80693 If **perm** is not specified, the '=' operation shall make no further modifications to the file
80694 mode bits.

80695 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
80696 other permissions, except for those with corresponding bits in the file mode creation mask
80697 of the invoking process, shall be set.

80698 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

80699 When using the symbolic mode form on a regular file, it is implementation-defined whether or
80700 not:

- 80701 • Requests to set the set-user-ID-on-execution or set-group-ID-on-execution bit when all
80702 execute bits are currently clear and none are being set are ignored.
- 80703 • Requests to clear all execute bits also clear the set-user-ID-on-execution and set-group-ID-
80704 on-execution bits.
- 80705 • Requests to clear the set-user-ID-on-execution or set-group-ID-on-execution bits when all
80706 execute bits are currently clear are ignored. However, if the command *ls -l file* writes an *s*
80707 in the position indicating that the set-user-ID-on-execution or set-group-ID-on-execution is
80708 set, the commands *chmod u-s file* or *chmod g-s file*, respectively, shall not be ignored.

80709 When using the symbolic mode form on other file types, it is implementation-defined whether
80710 or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are
80711 honored.

80712 If the **who** symbol **o** is used in conjunction with the **perm** symbol **s** with no other **who** symbols
80713 being specified, the set-user-ID-on-execution and set-group-ID-on-execution bits shall not be
80714 modified. It shall not be an error to specify the **who** symbol **o** in conjunction with the **perm**
80715 symbol **s**.

80716 XSI The **perm** symbol **t** shall specify the **S_ISVTX** bit. When used with a file of type directory, it can
80717 be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to specify a **who**
80718 symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning of these
80719 combinations is unspecified. The effect when using the **perm** symbol **t** with any file type other
80720 than directory is unspecified.

80721 For an octal integer *mode* operand, the file mode bits shall be set absolutely.

80722 For each bit set in the octal number, the corresponding file permission bit shown in the following
80723 table shall be set; all other file permission bits shall be cleared. For regular files, for each bit set in
80724 the octal number corresponding to the set-user-ID-on-execution or the set-group-ID-on-
80725 execution, bits shown in the following table shall be set; if these bits are not set in the octal
80726 number, they are cleared. For other file types, it is implementation-defined whether or not
80727 requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are
80728 honored.

Octal	Mode Bit	Octal	Mode Bit	Octal	Mode Bit	Octal	Mode Bit
4000	S_ISUID	0400	S_IRUSR	0040	S_IRGRP	0004	S_IROTH
2000	S_ISGID	0200	S_IWUSR	0020	S_IWGRP	0002	S_IWOTH
1000	S_ISVTX	0100	S_IXUSR	0010	S_IXGRP	0001	S_IXOTH

80729
80730
80731
80732 XSI
80733 When bits are set in the octal number other than those listed in the table above, the behavior is
80734 unspecified.

Grammar for chmod

The grammar and lexical conventions in this section describe the syntax for the *symbolic_mode* operand. The general conventions for this style of grammar are described in [Section 1.3](#) (on page 2287). A valid *symbolic_mode* can be represented as the non-terminal symbol *symbolic_mode* in the grammar. This formal syntax shall take precedence over the preceding text syntax description.

The lexical processing is based entirely on single characters. Implementations need not allow <blank> characters within the single argument being processed.

```
%start    symbolic_mode
%%

symbolic_mode    : clause
                  | symbolic_mode ',' clause
                  ;

clause           : actionlist
                  | wholist actionlist
                  ;

wholist          : who
                  | wholist who
                  ;

who              : 'u' | 'g' | 'o' | 'a'
                  ;

actionlist       : action
                  | actionlist action
                  ;

action           : op
                  | op permlist
                  | op permcopy
                  ;

permcopy         : 'u' | 'g' | 'o'
                  ;

op               : '+' | '-' | '='
                  ;

permlist         : perm
                  | perm permlist
                  ;

perm             : 'r' | 'w' | 'x' | 'X' | 's' | 't'
```

EXIT STATUS

The following exit values shall be returned:

- 0 The utility executed successfully and all requested changes were made.
- >0 An error occurred.

chmod

80776 CONSEQUENCES OF ERRORS
80777 Default.

80778 APPLICATION USAGE

80779 Some implementations of the *chmod* utility change the mode of a directory before the files in the
80780 directory when performing a recursive (**-R** option) change; others change the directory mode
80781 after the files in the directory. If an application tries to remove read or search permission for a
80782 file hierarchy, the removal attempt fails if the directory is changed first; on the other hand, trying
80783 to re-enable permissions to a restricted hierarchy fails if directories are changed last. Users
80784 should not try to make a hierarchy inaccessible to themselves.

80785 Some implementations of *chmod* never used the *umask* of the process when changing modes;
80786 systems conformant with this volume of POSIX.1-2008 do so when **who** is not specified. Note
80787 the difference between:

80788 `chmod a-w file`

80789 which removes all write permissions, and:

80790 `chmod -- -w file`

80791 which removes write permissions that would be allowed if **file** was created with the same
80792 *umask*.

80793 Conforming applications should never assume that they know how the set-user-ID and set-
80794 group-ID bits on directories are interpreted.

80795 EXAMPLES

Mode	Results
<i>a+=</i>	Equivalent to <i>a+,a=</i> ; clears all file mode bits.
<i>go+-w</i>	Equivalent to <i>go+,go-w</i> ; clears group and other write bits.
<i>g=o-w</i>	Equivalent to <i>g=o,g-w</i> ; sets group bit to match other bits and then clears group write bit.
<i>g-r+w</i>	Equivalent to <i>g-r,g+w</i> ; clears group read bit and sets group write bit.
<i>uo=g</i>	Sets owner bits to match group bits and sets other bits to match group bits.

80806 RATIONALE

80807 The functionality of *chmod* is described substantially through references to concepts defined in
80808 the System Interfaces volume of POSIX.1-2008. In this way, there is less duplication of effort
80809 required for describing the interactions of permissions. However, the behavior of this utility is
80810 not described in terms of the *chmod()* function from the System Interfaces volume of
80811 POSIX.1-2008 because that specification requires certain side-effects upon alternate file access
80812 control mechanisms that might not be appropriate, depending on the implementation.

80813 Implementations that support mandatory file and record locking as specified by the 1984
80814 /usr/group standard historically used the combination of set-group-ID bit set and group
80815 execute bit clear to indicate mandatory locking. This condition is usually set or cleared with the
80816 symbolic mode **perm** symbol **l** instead of the **perm** symbols **s** and **x** so that the mandatory
80817 locking mode is not changed without explicit indication that that was what the user intended.
80818 Therefore, the details on how the implementation treats these conditions must be defined in the
80819 documentation. This volume of POSIX.1-2008 does not require mandatory locking (nor does the
80820 System Interfaces volume of POSIX.1-2008), but does allow it as an extension. However, this
80821 volume of POSIX.1-2008 does require that the *ls* and *chmod* utilities work consistently in this

area. If *ls -l file* indicates that the set-group-ID bit is set, *chmod g-s file* must clear it (assuming appropriate privileges exist to change modes).

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. This problem is avoided here by specifying only 0 and >0 as exit values.

The System Interfaces volume of POSIX.1-2008 indicates that implementation-defined restrictions may cause the S_ISUID and S_ISGID bits to be ignored. This volume of POSIX.1-2008 allows the *chmod* utility to choose to modify these bits before calling *chmod()* (or some function providing equivalent capabilities) for non-regular files. Among other things, this allows implementations that use the set-user-ID and set-group-ID bits on directories to enable extended features to handle these extensions in an intelligent manner.

The **X perm** symbol was adopted from BSD-based systems because it provides commonly desired functionality when doing recursive (**-R** option) modifications. Similar functionality is not provided by the *find* utility. Historical BSD versions of *chmod*, however, only supported **X** with *op+*; it has been extended in this volume of POSIX.1-2008 because it is also useful with *op=*. (It has also been added for *op-* even though it duplicates **x**, in this case, because it is intuitive and easier to explain.)

The grammar was extended with the *permcop* non-terminal to allow historical-practice forms of symbolic modes like **o=u -g** (that is, set the “other” permissions to the permissions of “owner” minus the permissions of “group”).

FUTURE DIRECTIONS

None.

SEE ALSO

ls, umask

XBD [Section 4.4](#) (on page 108), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

XSH *chmod()*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- Octal modes have been kept and made mandatory despite being marked obsolescent in the ISO POSIX-2: 1993 standard.

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section to “Default.”.

The Open Group Base Resolution bwg2001-010 is applied, adding the description of the S_ISVTX bit and the **t perm** symbol as part of the XSI option.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/16 is applied, changing the XSI shaded text in the EXTENDED DESCRIPTION from:

“The **perm** symbol **t** shall specify the S_ISVTX bit and shall apply to directories only. The effect when using it with any other file type is unspecified. It can be used with the **who** symbols **o**, **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u** or **g** in conjunction with the **perm** symbol **t**; it shall be ignored for **u** and **g**.”

chmod*Utilities*

80866 to:

80867 “The **perm** symbol **t** shall specify the S_ISVTX bit. When used with a file of type directory,

80868 it can be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to

80869 specify a **who** symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning

80870 of these combinations is unspecified. The effect when using the **perm** symbol **t** with any

80871 file type other than directory is unspecified.”

80872 This change is to permit historical behavior.

80873 **Issue 7**

80874 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80875 Austin Group Interpretation 1003.1-2001 #130 is applied, adding text to the DESCRIPTION

80876 about about marking for update the last file status change timestamp of the file.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

chown**NAME**

chown — change the file ownership

SYNOPSIS

chown [-h] *owner[:group]* *file...*

chown -R [-H|-L|-P] *owner[:group]* *file...*

DESCRIPTION

The *chown* utility shall set the user ID of the file named by each *file* operand to the user ID specified by the *owner* operand.

For each *file* operand, or, if the **-R** option is used, each file encountered while walking the directory trees specified by the *file* operands, the *chown* utility shall perform actions equivalent to the *chown()* function defined in the System Interfaces volume of POSIX.1-2008, called with the following arguments:

1. The *file* operand shall be used as the *path* argument.
2. The user ID indicated by the *owner* portion of the first operand shall be used as the *owner* argument.
3. If the *group* portion of the first operand is given, the group ID indicated by it shall be used as the *group* argument; otherwise, the group ownership shall not be changed.

Unless *chown* is invoked by a process with appropriate privileges, the set-user-ID and set-group-ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-group-ID bits of other file types may be cleared.

OPTIONS

The *chown* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported by the implementation:

- h** For each file operand that names a file of type symbolic link, *chown* shall attempt to set the user ID of the symbolic link. If a group ID was specified, for each file operand that names a file of type symbolic link, *chown* shall attempt to set the group ID of the symbolic link.
- H** If the **-R** option is specified and a symbolic link referencing a file of type directory is specified on the command line, *chown* shall change the user ID (and group ID, if specified) of the directory referenced by the symbolic link and all files in the file hierarchy below it.
- L** If the **-R** option is specified and a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, *chown* shall change the user ID (and group ID, if specified) of the directory referenced by the symbolic link and all files in the file hierarchy below it.
- P** If the **-R** option is specified and a symbolic link is specified on the command line or encountered during the traversal of a file hierarchy, *chown* shall change the owner ID (and group ID, if specified) of the symbolic link. The *chown* utility shall not follow the symbolic link to any other part of the file hierarchy.
- R** Recursively change file user and group IDs. For each *file* operand that names a directory, *chown* shall change the user ID (and group ID, if specified) of the directory and all files in the file hierarchy below it. Unless a **-H**, **-L**, or **-P** option is specified, it is unspecified which of these options will be used as the default.

Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be

chown*Utilities*

80921 considered an error. The last option specified shall determine the behavior of the utility.

OPERANDS

80922 The following operands shall be supported:

80923 *owner[:group]* A user ID and optional group ID to be assigned to *file*. The *owner* portion of this operand shall be a user name from the user database or a numeric user ID. Either specifies a user ID which shall be given to each file named by one of the *file* operands. If a numeric *owner* operand exists in the user database as a user name, the user ID number associated with that user name shall be used as the user ID. Similarly, if the *group* portion of this operand is present, it shall be a group name from the group database or a numeric group ID. Either specifies a group ID which shall be given to each file. If a numeric group operand exists in the group database as a group name, the group ID number associated with that group name shall be used as the group ID.

80934 *file* A pathname of a file whose user ID is to be modified.

STDIN

80935 Not used.

INPUT FILES

80936 None.

ENVIRONMENT VARIABLES

80937 The following environment variables shall affect the execution of *chown*:

80941 *LANG* Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

80944 *LC_ALL* If set to a non-empty string value, override the values of all the other internationalization variables.

80946 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

80949 *LC_MESSAGES* Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

80952 *XSI NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

80953 Default.

STDOUT

80954 Not used.

STDERR

80955 The standard error shall be used only for diagnostic messages.

OUTPUT FILES

80956 None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Only the owner of a file or the user with appropriate privileges may change the owner or group of a file.

Some implementations restrict the use of *chown* to a user with appropriate privileges.

EXAMPLES

None.

RATIONALE

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. These are masked by specifying only 0 and >0 as exit values.

The functionality of *chown* is described substantially through references to functions in the System Interfaces volume of POSIX.1-2008. In this way, there is no duplication of effort required for describing the interactions of permissions, multiple groups, and so on.

The 4.3 BSD method of specifying both owner and group was included in this volume of POSIX.1-2008 because:

- There are cases where the desired end condition could not be achieved using the *chgrp* and *chown* (that only changed the user ID) utilities. (If the current owner is not a member of the desired group and the desired owner is not a member of the current group, the *chown()* function could fail unless both owner and group are changed at the same time.)
- Even if they could be changed independently, in cases where both are being changed, there is a 100% performance penalty caused by being forced to invoke both utilities.

The BSD syntax *user[group]* was changed to *user[:group]* in this volume of POSIX.1-2008 because the <period> is a valid character in login names (as specified by the Base Definitions volume of POSIX.1-2008, login names consist of characters in the portable filename character set). The <colon> character was chosen as the replacement for the <period> character because it would never be allowed as a character in a user name or group name on historical implementations.

The **-R** option is considered by some observers as an undesirable departure from the historical UNIX system tools approach; since a tool, *find*, already exists to recurse over directories, there seemed to be no good reason to require other tools to have to duplicate that functionality. However, the **-R** option was deemed an important user convenience, is far more efficient than forking a separate process for each element of the directory hierarchy, and is in widespread historical use.

chown

Utilities

81002 **FUTURE DIRECTIONS**

81003 None.

81004 **SEE ALSO**81005 *chgrp, chmod*81006 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)81007 XSH *chown()*81008 **CHANGE HISTORY**

81009 First released in Issue 2.

81010 **Issue 6**81011 New options **-h**, **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These
81012 options affect the processing of symbolic links.

81013 The normative text is reworded to avoid use of the term “must” for application requirements.

81014 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS
81015 section to “Default.”.81016 The “otherwise, ...” text in item 3. of the DESCRIPTION is changed to “otherwise, the group
81017 ownership shall not be changed”.81018 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/17 is applied, changing the SYNOPSIS to
81019 make it clear that **-h** and **-R** are optional.81020 **Issue 7**81021 SD5-XCU-ERN-9 is applied, removing the **-R** from the first line of the SYNOPSIS.

81022 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

81023 The description of the **-h** and **-P** options is revised.

Utilities

cksum

NAME

cksum — write file checksums and sizes

SYNOPSIS

cksum [*file*...]

DESCRIPTION

The *cksum* utility shall calculate and write to standard output a cyclic redundancy check (CRC) for each input file, and also write to standard output the number of octets in each file. The CRC used is based on the polynomial used for CRC error checking in the ISO/IEC 8802-3:1996 standard (Ethernet).

The encoding for the CRC checksum is defined by the generating polynomial:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Mathematically, the CRC value corresponding to a given file shall be defined by the following procedure:

1. The n bits to be evaluated are considered to be the coefficients of a mod 2 polynomial $M(x)$ of degree $n-1$. These n bits are the bits from the file, with the most significant bit being the most significant bit of the first octet of the file and the last bit being the least significant bit of the last octet, padded with zero bits (if necessary) to achieve an integral number of octets, followed by one or more octets representing the length of the file as a binary value, least significant octet first. The smallest number of octets capable of representing this integer shall be used.
2. $M(x)$ is multiplied by x^{32} (that is, shifted left 32 bits) and divided by $G(x)$ using mod 2 division, producing a remainder $R(x)$ of degree ≤ 31 .
3. The coefficients of $R(x)$ are considered to be a 32-bit sequence.
4. The bit sequence is complemented and the result is the CRC.

OPTIONS

None.

OPERANDS

The following operand shall be supported:

file A pathname of a file to be checked. If no *file* operands are specified, the standard input shall be used.

STDIN

The standard input shall be used if no *file* operands are specified, and shall be used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

INPUT FILES

The input files can be any file type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *cksum*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

cksum

Utilities

81067	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
81068		
81069		
81070	<i>LC_MESSAGES</i>	
81071		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
81072		
81073	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
81074	ASYNCHRONOUS EVENTS	
81075	Default.	
81076	STDOUT	
81077	For each file processed successfully, the <i>cksum</i> utility shall write in the following format:	
81078	"%u %d %s\n", <checksum>, <# of octets>, <pathname>	
81079	If no <i>file</i> operand was specified, the pathname and its leading <space> shall be omitted.	
81080	STDERR	
81081	The standard error shall be used only for diagnostic messages.	
81082	OUTPUT FILES	
81083	None.	
81084	EXTENDED DESCRIPTION	
81085	None.	
81086	EXIT STATUS	
81087	The following exit values shall be returned:	
81088	0	All files were processed successfully.
81089	>0	An error occurred.
81090	CONSEQUENCES OF ERRORS	
81091	Default.	
81092	APPLICATION USAGE	
81093	The <i>cksum</i> utility is typically used to quickly compare a suspect file against a trusted version of the same, such as to ensure that files transmitted over noisy media arrive intact. However, this comparison cannot be considered cryptographically secure. The chances of a damaged file producing the same CRC as the original are small; deliberate deception is difficult, but probably not impossible.	
81094		
81095		
81096		
81097		
81098	Although input files to <i>cksum</i> can be any type, the results need not be what would be expected on character special device files or on file types not described by the System Interfaces volume of POSIX.1-2008. Since this volume of POSIX.1-2008 does not specify the block size used when doing input, checksums of character special files need not process all of the data in those files.	
81099		
81100		
81101		
81102	The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted between two systems and undergoes any data transformation (such as changing little-endian byte ordering to big-endian), identical CRC values cannot be expected. Implementations performing such transformations may extend <i>cksum</i> to handle such situations.	
81103		
81104		
81105		
81106	EXAMPLES	
81107	None.	

RATIONALE

The following C-language program can be used as a model to describe the algorithm. It assumes that a **char** is one octet. It also assumes that the entire file is available for one pass through the function. This was done for simplicity in demonstrating the algorithm, rather than as an implementation model.

```
static unsigned long crctab[] = {
0x00000000,
0x04c11db7, 0x09823b6e, 0x0d4326d9, 0x130476dc, 0x17c56b6b,
0x1a864db2, 0x1e475005, 0x2608edb8, 0x22c9f00f, 0x2f8ad6d6,
0x2b4bcb61, 0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbd8d,
0x4c11db70, 0x48d0c6c7, 0x4593e01e, 0x4152fda9, 0x5f15adac,
0x5bd4b01b, 0x569796c2, 0x52568b75, 0x6a1936c8, 0x6ed82b7f,
0x639b0da6, 0x675a1011, 0x791d4014, 0x7ddc5da3, 0x709f7b7a,
0x745e66cd, 0x9823b6e0, 0x9ce2ab57, 0x91a18d8e, 0x95609039,
0x8b27c03c, 0x8fe6dd8b, 0x82a5fb52, 0x8664e6e5, 0xbe2b5b58,
0xbaea46ef, 0xb7a96036, 0xb3687d81, 0xad2f2d84, 0xa9ee3033,
0xa4ad16ea, 0xa06c0b5d, 0xd4326d90, 0xd0f37027, 0xddb056fe,
0xd9714b49, 0xc7361b4c, 0xc3f706fb, 0xceb42022, 0xca753d95,
0xf23a8028, 0xf6fb9d9f, 0xfbb8bb46, 0xff79a6f1, 0xe13ef6f4,
0xe5fdeb43, 0xe8bccd9a, 0xec7dd02d, 0x34867077, 0x30476dc0,
0x3d044b19, 0x39c556ae, 0x278206ab, 0x23431b1c, 0x2e003dc5,
0x2ac12072, 0x128e9dcf, 0x164f8078, 0x1b0ca6a1, 0x1fcd8bb16,
0x018aeb13, 0x054bf6a4, 0x0808d07d, 0x0cc9cdca, 0x7897ab07,
0x7c56b6b0, 0x71159069, 0x75d48dde, 0x6b93dddb, 0x6f52c06c,
0x6211e6b5, 0x66d0fb02, 0x5e9f46bf, 0x5a5e5b08, 0x571d7dd1,
0x53dc6066, 0x4d9b3063, 0x495a2dd4, 0x44190b0d, 0x40d816ba,
0xaca5c697, 0xa864db20, 0xa527fd9, 0xa1e6e04e, 0xbfa1b04b,
0xbb60adfc, 0xb6238b25, 0xb2e29692, 0x8aad2b2f, 0x8e6c3698,
0x832f1041, 0x87ee0df6, 0x99a95df3, 0x9d684044, 0x902b669d,
0x94ea7b2a, 0xe0b41de7, 0xe4750050, 0xe9362689, 0xedf73b3e,
0xf3b06b3b, 0xf771768c, 0xfa325055, 0xfef34de2, 0xc6bcf05f,
0xc27dede8, 0xcf3ecb31, 0xcbffd686, 0xd5b88683, 0xd1799b34,
0xdc3abded, 0xd8fba05a, 0x690ce0ee, 0x6dcd9d59, 0x608edb80,
0x644fc637, 0xa089632, 0x7ec98b85, 0x738aad5c, 0x774bb0eb,
0x4f040d56, 0x4bc510e1, 0x46863638, 0x42472b8f, 0x5c007b8a,
0x58c1663d, 0x558240e4, 0x51435d53, 0x251d3b9e, 0x21dc2629,
0x2c9f00f0, 0x285e1d47, 0x36194d42, 0x32d850f5, 0x3f9b762c,
0x3b5a6b9b, 0x0315d626, 0x07d4cb91, 0x0a97ed48, 0x0e56f0ff,
0x1011a0fa, 0x14d0bd4d, 0x19939b94, 0x1d528623, 0xf12f560e,
0xf5ee4bb9, 0xf8ad6d60, 0xfc6c70d7, 0xe22b20d2, 0xe6ea3d65,
0xeba91bbc, 0xef68060b, 0xd727bbb6, 0xd3e6a601, 0xdea580d8,
0xda649d6f, 0xc423cd6a, 0xc0e2d0dd, 0xcda1f604, 0xc960ebb3,
0xbd3e8d7e, 0xb9ff90c9, 0xb4bcb610, 0xb07daba7, 0xae3afba2,
0xaafbe615, 0xa7b8c0cc, 0xa379dd7b, 0x9b3660c6, 0x9ff77d71,
0x92b45ba8, 0x9675461f, 0x8832161a, 0x8cf30bad, 0x81b02d74,
0x857130c3, 0x5d8a9099, 0x594b8d2e, 0x5408abf7, 0x50c9b640,
0x4e8ee645, 0x4a4ffb2, 0x470cdd2b, 0x43cdc09c, 0x7b827d21,
0x7f436096, 0x7200464f, 0x76c15bf8, 0x68860bfd, 0x6c47164a,
0x61043093, 0x65c52d24, 0x119b4be9, 0x155a565e, 0x18197087,
0x1cd86d30, 0x029f3d35, 0x065e2082, 0x0b1d065b, 0x0fdc1bec,
0x3793a651, 0x3352bbe6, 0x3e119d3f, 0x3ad08088, 0x2497d08d,
```

```

81159      0x2056cd3a, 0x2d15ebe3, 0x29d4f654, 0xc5a92679, 0xc1683bce,
81160      0xcc2b1d17, 0xc8ea00a0, 0xd6ad50a5, 0xd26c4d12, 0xdf2f6bcb,
81161      0xdbee767c, 0xe3a1c1bc1, 0xe760d676, 0xea23f0af, 0xee2ed18,
81162      0xf0a5bd1d, 0xf464a0aa, 0xf9278673, 0xfde69bc4, 0x89b8fd09,
81163      0x8d79e0be, 0x803ac667, 0x84fbd0bd0, 0x9abc8bd5, 0x9e7d9662,
81164      0x933eb0bb, 0x97ffad0c, 0xafb010b1, 0xab710d06, 0xa6322bdf,
81165      0xa2f33668, 0xbcb4666d, 0xb8757bda, 0xb5365d03, 0xb1f740b4
81166      };

81167      unsigned long memcrc(const unsigned char *b, size_t n)
81168      {
81169      /*   Input arguments:
81170       *   const char*   b == byte sequence to checksum
81171       *   size_t        n == length of sequence
81172       */

81173          register unsigned   i, c, s = 0;

81174          for (i = n; i > 0; --i) {
81175              c = (unsigned) (*b++);
81176              s = (s << 8) ^ crctab[(s >> 24) ^ c];
81177          }

81178          /* Extend with the length of the string. */
81179          while (n != 0) {
81180              c = n & 0377;
81181              n >>= 8;
81182              s = (s << 8) ^ crctab[(s >> 24) ^ c];
81183          }

81184          return ~s;
81185      }

```

The historical practice of writing the number of “blocks” has been changed to writing the number of octets, since the latter is not only more useful, but also since historical implementations have not been consistent in defining what a “block” meant.

The algorithm used was selected to increase the operational robustness of *cksum*. Neither the System V nor BSD *sum* algorithm was selected. Since each of these was different and each was the default behavior on those systems, no realistic compromise was available if either were selected—some set of historical applications would break. Therefore, the name was changed to *cksum*. Although the historical *sum* commands will probably continue to be provided for many years, programs designed for portability across systems should use the new name.

The algorithm selected is based on that used by the ISO/IEC 8802-3:1996 standard (Ethernet) for the frame check sequence field. The algorithm used does not match the technical definition of a *checksum*; the term is used for historical reasons. The length of the file is included in the CRC calculation because this parallels inclusion of a length field by Ethernet in its CRC, but also because it guards against inadvertent collisions between files that begin with different series of zero octets. The chance that two different files produce identical CRCs is much greater when their lengths are not considered. Keeping the length and the checksum of the file itself separate would yield a slightly more robust algorithm, but historical usage has always been that a single number (the checksum as printed) represents the signature of the file. It was decided that historical usage was the more important consideration.

Early proposals contained modifications to the Ethernet algorithm that involved extracting table

values whenever an intermediate result became zero. This was demonstrated to be less robust than the current method and mathematically difficult to describe or justify.

The calculation used is identical to that given in pseudo-code in the referenced Sarwate article. The pseudo-code rendition is:

```
X <- 0; Y <- 0;
for i <- m -1 step -1 until 0 do
  begin
    T <- X(1) ^ A[i];
    X(1) <- X(0); X(0) <- Y(1); Y(1) <- Y(0); Y(0) <- 0;
    comment: f[T] and f'[T] denote the T-th words in the
             table f and f' ;
    X <- X ^ f[T]; Y <- Y ^ f'[T];
  end
```

The pseudo-code is reproduced exactly as given; however, note that in the case of *cksum*, *A[i]* represents a byte of the file, the words *X* and *Y* are treated as a single 32-bit value, and the tables *f* and *f'* are a single table containing 32-bit values.

The referenced Sarwate article also discusses generating the table.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD [Chapter 8](#) (on page 173)

CHANGE HISTORY

First released in Issue 4.

Issue 7

Austin Group Interpretation 1003.1-2001 #092 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

cmp

Utilities

81232 **NAME**81233 `cmp` — compare two files81234 **SYNOPSIS**81235 `cmp [-l|-s] file1 file2`81236 **DESCRIPTION**

81237 The *cmp* utility shall compare two files. The *cmp* utility shall write no output if the files are the
 81238 same. Under default options, if they differ, it shall write to standard output the byte and line
 81239 number at which the first difference occurred. Bytes and lines shall be numbered beginning with
 81240 1.

81241 **OPTIONS**81242 The *cmp* utility shall conform to XBD Section 12.2 (on page 215).

81243 The following options shall be supported:

81244 **-l** (Lowercase ell.) Write the byte number (decimal) and the differing bytes (octal) for
 81245 each difference.

81246 **-s** Write nothing for differing files; return exit status only

81247 **OPERANDS**

81248 The following operands shall be supported:

81249 *file1* A pathname of the first file to be compared. If *file1* is '-', the standard input shall
 81250 be used.

81251 *file2* A pathname of the second file to be compared. If *file2* is '-', the standard input
 81252 shall be used.

81253 If both *file1* and *file2* refer to standard input or refer to the same FIFO special, block special, or
 81254 character special file, the results are undefined.

81255 **STDIN**

81256 The standard input shall be used only if the *file1* or *file2* operand refers to standard input. See the
 81257 INPUT FILES section.

81258 **INPUT FILES**

81259 The input files can be any file type.

81260 **ENVIRONMENT VARIABLES**81261 The following environment variables shall affect the execution of *cmp*:

81262 **LANG** Provide a default value for the internationalization variables that are unset or null.
 81263 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 81264 variables used to determine the values of locale categories.)

81265 **LC_ALL** If set to a non-empty string value, override the values of all the other
 81266 internationalization variables.

81267 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 81268 characters (for example, single-byte as opposed to multi-byte characters in
 81269 arguments).

81270 **LC_MESSAGES**

81271 Determine the locale that should be used to affect the format and contents of
 81272 diagnostic messages written to standard error and informative messages written to
 81273 standard output.

81274 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

81275 ASYNCHRONOUS EVENTS

81276 Default.

81277 STDOUT

81278 In the POSIX locale, results of the comparison shall be written to standard output. When no
81279 options are used, the format shall be:

81280 "%s %s differ: char %d, line %d\n", *file1*, *file2*,
81281 <byte number>, <line number>

81282 When the **-l** option is used, the format shall be:

81283 "%d %o %o\n", <byte number>, <differing byte>,
81284 <differing byte>

81285 for each byte that differs. The first <differing byte> number is from *file1* while the second is from
81286 *file2*. In both cases, <byte number> shall be relative to the beginning of the file, beginning with 1.

81287 No output shall be written to standard output when the **-s** option is used.

81288 STDERR

81289 The standard error shall be used only for diagnostic messages. If the **-l** option is used and *file1*
81290 and *file2* differ in length, or if the **-s** option is not used and *file1* and *file2* are identical for the
81291 entire length of the shorter file, in the POSIX locale the following diagnostic message shall be
81292 written:

81293 "cmp: EOF on %s%s\n", <name of shorter file>, <additional info>

81294 The <additional info> field shall either be null or a string that starts with a <blank> and contains
81295 no <newline> characters. Some implementations report on the number of lines in this case.

81296 OUTPUT FILES

81297 None.

81298 EXTENDED DESCRIPTION

81299 None.

81300 EXIT STATUS

81301 The following exit values shall be returned:

81302 0 The files are identical.

81303 1 The files are different; this includes the case where one file is identical to the first part of the
81304 other.

81305 >1 An error occurred.

81306 CONSEQUENCES OF ERRORS

81307 Default.

APPLICATION USAGE

Although input files to *cmp* can be any type, the results might not be what would be expected on character special device files or on file types not described by the System Interfaces volume of POSIX.1-2008. Since this volume of POSIX.1-2008 does not specify the block size used when doing input, comparisons of character special files need not compare all of the data in those files.

For files which are not text files, line numbers simply reflect the presence of a <newline>, without any implication that the file is organized into lines.

EXAMPLES

None.

RATIONALE

The global language in [Section 1.4](#) (on page 2288) indicates that using two mutually-exclusive options together produces unspecified results. Some System V implementations consider the option usage:

```
cmp -l -s ...
```

to be an error. They also treat:

```
cmp -s -l ...
```

as if no options were specified. Both of these behaviors are considered bugs, but are allowed.

The word **char** in the standard output format comes from historical usage, even though it is actually a byte number. When *cmp* is supported in other locales, implementations are encouraged to use the word *byte* or its equivalent in another language. Users should not interpret this difference to indicate that the functionality of the utility changed between locales.

Some implementations report on the number of lines in the identical-but-shorter file case. This is allowed by the inclusion of the <additional info> fields in the output format. The restriction on having a leading <blank> and no <newline> characters is to make parsing for the filename easier. It is recognized that some filenames containing white-space characters make parsing difficult anyway, but the restriction does aid programs used on systems where the names are predominantly well behaved.

FUTURE DIRECTIONS

None.

SEE ALSO

comm, *diff*

XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 7

SD5-XCU-ERN-96 is applied, updating the STDERR section.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Utilities

comm

NAME

comm — select or reject lines common to two files

SYNOPSIS

comm [-123] file1 file2

DESCRIPTION

The *comm* utility shall read *file1* and *file2*, which should be ordered in the current collating sequence, and produce three text columns as output: lines only in *file1*, lines only in *file2*, and lines in both files.

If the lines in both files are not ordered according to the collating sequence of the current locale, the results are unspecified.

OPTIONS

The *comm* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

- 1 Suppress the output column of lines unique to *file1*.
- 2 Suppress the output column of lines unique to *file2*.
- 3 Suppress the output column of lines duplicated in *file1* and *file2*.

OPERANDS

The following operands shall be supported:

- file1* A pathname of the first file to be compared. If *file1* is '-', the standard input shall be used.
- file2* A pathname of the second file to be compared. If *file2* is '-', the standard input shall be used.

If both *file1* and *file2* refer to standard input or to the same FIFO special, block special, or character special file, the results are undefined.

STDIN

The standard input shall be used only if one of the *file1* or *file2* operands refers to standard input. See the INPUT FILES section.

INPUT FILES

The input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *comm*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE

Determine the locale for the collating sequence *comm* expects to have been used when the input files were sorted.

LC_CTYPE

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

- 81387 *LC_MESSAGES*
 81388 Determine the locale that should be used to affect the format and contents of
 81389 diagnostic messages written to standard error.
- 81390 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 81391 **ASYNCHRONOUS EVENTS**
 81392 Default.
- 81393 **STDOUT**
 81394 The *comm* utility shall produce output depending on the options selected. If the *-1*, *-2*, and *-3*
 81395 options are all selected, *comm* shall write nothing to standard output.
- 81396 If the *-1* option is not selected, lines contained only in *file1* shall be written using the format:
 81397 "%s\n", <line in file1>
- 81398 If the *-2* option is not selected, lines contained only in *file2* are written using the format:
 81399 "%s%s\n", <lead>, <line in file2>
- 81400 where the string <lead> is as follows:
 81401 <tab> The *-1* option is not selected.
 81402 null string The *-1* option is selected.
- 81403 If the *-3* option is not selected, lines contained in both files shall be written using the format:
 81404 "%s%s\n", <lead>, <line in both>
- 81405 where the string <lead> is as follows:
 81406 <tab><tab> Neither the *-1* nor the *-2* option is selected.
 81407 <tab> Exactly one of the *-1* and *-2* options is selected.
 81408 null string Both the *-1* and *-2* options are selected.
- 81409 If the input files were ordered according to the collating sequence of the current locale, the lines
 81410 written shall be in the collating sequence of the original lines.
- 81411 **STDERR**
 81412 The standard error shall be used only for diagnostic messages.
- 81413 **OUTPUT FILES**
 81414 None.
- 81415 **EXTENDED DESCRIPTION**
 81416 None.
- 81417 **EXIT STATUS**
 81418 The following exit values shall be returned:
 81419 0 All input files were successfully output as specified.
 81420 >0 An error occurred.
- 81421 **CONSEQUENCES OF ERRORS**
 81422 Default.

APPLICATION USAGE

If the input files are not properly presorted, the output of *comm* might not be useful.

EXAMPLES

If a file named **xcu** contains a sorted list of the utilities in this volume of POSIX.1-2008, a file named **xpg3** contains a sorted list of the utilities specified in the X/Open Portability Guide, Issue 3, and a file named **svid89** contains a sorted list of the utilities in the System V Interface Definition Third Edition:

```
comm -23 xcu xpg3 | comm -23 - svid89
```

would print a list of utilities in this volume of POSIX.1-2008 not specified by either of the other documents:

```
comm -12 xcu xpg3 | comm -12 - svid89
```

would print a list of utilities specified by all three documents, and:

```
comm -12 xpg3 svid89 | comm -23 - xcu
```

would print a list of utilities specified by both XPG3 and the SVID, but not specified in this volume of POSIX.1-2008.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cmp, *diff*, *sort*, *uniq*

XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term “must” for application requirements.

command

Utilities

81449 **NAME**81450 `command` — execute a simple command81451 **SYNOPSIS**81452 `command [-p] command_name [argument...]`81453 `command [-p] [-v|-V] command_name`81454 **DESCRIPTION**81455 The *command* utility shall cause the shell to treat the arguments as a simple command,
81456 suppressing the shell function lookup that is described in [Section 2.9.1.1](#) (on page 2317), item 1b.81457 If the *command_name* is the same as the name of one of the special built-in utilities, the special
81458 properties in the enumerated list at the beginning of [Section 2.14](#) (on page 2334) shall not occur.
81459 In every other respect, if *command_name* is not the name of a function, the effect of *command*
81460 (with no options) shall be the same as omitting *command*.81461 When the `-v` or `-V` option is used, the *command* utility shall provide information concerning
81462 how a command name is interpreted by the shell.81463 **OPTIONS**81464 The *command* utility shall conform to XBD [Section 12.2](#) (on page 215).

81465 The following options shall be supported:

81466 **-p** Perform the command search using a default value for *PATH* that is guaranteed to
81467 find all of the standard utilities.81468 **-v** Write a string to standard output that indicates the pathname or command that
81469 will be used by the shell, in the current shell execution environment (see [Section](#)
81470 [2.12](#), on page 2331), to invoke *command_name*, but do not invoke *command_name*.81471 • Utilities, regular built-in utilities, *command_names* including a <slash>
81472 character, and any implementation-defined functions that are found using
81473 the *PATH* variable (as described in [Section 2.9.1.1](#), on page 2317), shall be
81474 written as absolute pathnames.81475 • Shell functions, special built-in utilities, regular built-in utilities not
81476 associated with a *PATH* search, and shell reserved words shall be written as
81477 just their names.81478 • An alias shall be written as a command line that represents its alias
81479 definition.81480 • Otherwise, no output shall be written and the exit status shall reflect that the
81481 name was not found.81482 **-V** Write a string to standard output that indicates how the name given in the
81483 *command_name* operand will be interpreted by the shell, in the current shell
81484 execution environment (see [Section 2.12](#), on page 2331), but do not invoke
81485 *command_name*. Although the format of this string is unspecified, it shall indicate
81486 in which of the following categories *command_name* falls and shall include the
81487 information stated:81488 • Utilities, regular built-in utilities, and any implementation-defined functions
81489 that are found using the *PATH* variable (as described in [Section 2.9.1.1](#), on
81490 page 2317), shall be identified as such and include the absolute pathname in
81491 the string.

Utilities

command

- 81492 • Other shell functions shall be identified as functions.
- 81493 • Aliases shall be identified as aliases and their definitions included in the
- 81494 string.
- 81495 • Special built-in utilities shall be identified as special built-in utilities.
- 81496 • Regular built-in utilities not associated with a *PATH* search shall be identified
- 81497 as regular built-in utilities. (The term “regular” need not be used.)
- 81498 • Shell reserved words shall be identified as reserved words.

81499 **OPERANDS**

81500 The following operands shall be supported:

81501 *argument* One of the strings treated as an argument to *command_name*.

81502 *command_name*

81503 The name of a utility or a special built-in utility.

81504 **STDIN**

81505 Not used.

81506 **INPUT FILES**

81507 None.

81508 **ENVIRONMENT VARIABLES**

81509 The following environment variables shall affect the execution of *command*:

81510 *LANG* Provide a default value for the internationalization variables that are unset or null.
 81511 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 81512 variables used to determine the values of locale categories.)

81513 *LC_ALL* If set to a non-empty string value, override the values of all the other
 81514 internationalization variables.

81515 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 81516 characters (for example, single-byte as opposed to multi-byte characters in
 81517 arguments).

81518 *LC_MESSAGES*

81519 Determine the locale that should be used to affect the format and contents of
 81520 diagnostic messages written to standard error and informative messages written to
 81521 standard output.

81522 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

81523 *PATH* Determine the search path used during the command search described in Section
 81524 2.9.1.1 (on page 2317), except as described under the *-p* option.

81525 **ASYNCHRONOUS EVENTS**

81526 Default.

81527 **STDOUT**

81528 When the *-v* option is specified, standard output shall be formatted as:

81529 "%s\n", *<pathname or command>*

81530 When the *-V* option is specified, standard output shall be formatted as:

81531 "%s\n", *<unspecified>*

command

Utilities

81532 **STDERR**

81533 The standard error shall be used only for diagnostic messages.

81534 **OUTPUT FILES**

81535 None.

81536 **EXTENDED DESCRIPTION**

81537 None.

81538 **EXIT STATUS**81539 When the `-v` or `-V` options are specified, the following exit values shall be returned:

81540 0 Successful completion.

81541 >0 The *command_name* could not be found or an error occurred.

81542 Otherwise, the following exit values shall be returned:

81543 126 The utility specified by *command_name* was found but could not be invoked.81544 127 An error occurred in the *command* utility or the utility specified by *command_name* could not
81545 be found.81546 Otherwise, the exit status of *command* shall be that of the simple command specified by the
81547 arguments to *command*.81548 **CONSEQUENCES OF ERRORS**

81549 Default.

81550 **APPLICATION USAGE**81551 The order for command search allows functions to override regular built-ins and path searches.
81552 This utility is necessary to allow functions that have the same name as a utility to call the utility
81553 (instead of a recursive call to the function).81554 The system default path is available using *getconf*; however, since *getconf* may need to have the
81555 *PATH* set up before it can be called itself, the following can be used:81556 `command -p getconf PATH`81557 There are some advantages to suppressing the special characteristics of special built-ins on
81558 occasion. For example:81559 `command exec > unwritable-file`81560 does not cause a non-interactive script to abort, so that the output status can be checked by the
81561 script.81562 The *command*, *env*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an
81563 error occurs so that applications can distinguish “failure to find a utility” from “invoked utility
81564 exited with an error indication”. The value 127 was chosen because it is not commonly used for
81565 other meanings; most utilities use small values for “normal error conditions” and the values
81566 above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen
81567 in a similar manner to indicate that the utility could be found, but not invoked. Some scripts
81568 produce meaningful error messages differentiating the 126 and 127 cases. The distinction
81569 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
81570 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
81571 any other reason.81572 Since the `-v` and `-V` options of *command* produce output in relation to the current shell execution
81573 environment, *command* is generally provided as a shell regular built-in. If it is called in a subshell
81574 or separate utility execution environment, such as one of the following:


```
81575 (PATH=foo command -v)
81576 nohup command -v
```

81577 it does not necessarily produce correct results. For example, when called with *nohup* or an *exec*
 81578 function, in a separate utility execution environment, most implementations are not able to
 81579 identify aliases, functions, or special built-ins.

81580 Two types of regular built-ins could be encountered on a system and these are described
 81581 separately by *command*. The description of command search in [Section 2.9.1.1](#) (on page 2317)
 81582 allows for a standard utility to be implemented as a regular built-in as long as it is found in the
 81583 appropriate place in a *PATH* search. So, for example, *command -v true* might yield */bin/true* or
 81584 some similar pathname. Other implementation-defined utilities that are not defined by this
 81585 volume of POSIX.1-2008 might exist only as built-ins and have no pathname associated with
 81586 them. These produce output identified as (regular) built-ins. Applications encountering these are
 81587 not able to count on *execing* them, using them with *nohup*, overriding them with a different
 81588 *PATH*, and so on.

81589 EXAMPLES

- 81590 1. Make a version of *cd* that always prints out the new working directory exactly once:

```
81591 cd() {
81592     command cd "$@" >/dev/null
81593     pwd
81594 }
```

- 81595 2. Start off a “secure shell script” in which the script avoids being spoofed by its parent:

```
81596 IFS='
81597 '
81598 # The preceding value should be <space><tab><newline>.
81599 # Set IFS to its default value.
81600 \unalias -a
81601 # Unset all possible aliases.
81602 # Note that unalias is escaped to prevent an alias
81603 # being used for unalias.
81604 unset -f command
81605 # Ensure command is not a user function.
81606 PATH="$(command -p getconf PATH):$PATH"
81607 # Put on a reliable PATH prefix.
81608 # ...
```

81609 At this point, given correct permissions on the directories called by *PATH*, the script has
 81610 the ability to ensure that any utility it calls is the intended one. It is being very cautious
 81611 because it assumes that implementation extensions may be present that would allow user
 81612 functions to exist when it is invoked; this capability is not specified by this volume of
 81613 POSIX.1-2008, but it is not prohibited as an extension. For example, the *ENV* variable
 81614 precedes the invocation of the script with a user start-up script. Such a script could define
 81615 functions to spoof the application.

81616 RATIONALE

81617 Since *command* is a regular built-in utility it is always found prior to the *PATH* search.

81618 There is nothing in the description of *command* that implies the command line is parsed any
 81619 differently from that of any other simple command. For example:

command

Utilities

command a | b ; c

is not parsed in any special way that causes ' | ' or ' ; ' to be treated other than a pipe operator or <semicolon> or that prevents function lookup on **b** or **c**.

The *command* utility is somewhat similar to the Eighth Edition shell *builtin* command, but since *command* also goes to the file system to search for utilities, the name *builtin* would not be intuitive.

The *command* utility is most likely to be provided as a regular built-in. It is not listed as a special built-in for the following reasons:

- The removal of exportable functions made the special precedence of a special built-in unnecessary.
- A special built-in has special properties (see [Section 2.14](#), on page 2334) that were inappropriate for invoking other utilities. For example, two commands such as:

```
date > unwritable-file
```

```
command date > unwritable-file
```

would have entirely different results; in a non-interactive script, the former would continue to execute the next command, the latter would abort. Introducing this semantic difference along with suppressing functions was seen to be non-intuitive.

The **-p** option is present because it is useful to be able to ensure a safe path search that finds all the standard utilities. This search might not be identical to the one that occurs through one of the *exec* functions (as defined in the System Interfaces volume of POSIX.1-2008) when *PATH* is unset. At the very least, this feature is required to allow the script to access the correct version of *getconf* so that the value of the default path can be accurately retrieved.

The *command* **-v** and **-V** options were added to satisfy requirements from users that are currently accomplished by three different historical utilities: *type* in the System V shell, *whence* in the KornShell, and *which* in the C shell. Since there is no historical agreement on how and what to accomplish here, the POSIX *command* utility was enhanced and the historical utilities were left unmodified. The C shell *which* merely conducts a path search. The KornShell *whence* is more elaborate—in addition to the categories required by POSIX, it also reports on tracked aliases, exported aliases, and undefined functions.

The output format of **-V** was left mostly unspecified because human users are its only audience. Applications should not be written to care about this information; they can use the output of **-v** to differentiate between various types of commands, but the additional information that may be emitted by the more verbose **-V** is not needed and should not be arbitrarily constrained in its verbosity or localization for application parsing reasons.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.9.1.1](#) (on page 2317), [Section 2.12](#) (on page 2331), [Section 2.14](#) (on page 2334), *sh*, *type*

XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

XSH *exec*

81660 **CHANGE HISTORY**

81661 First released in Issue 4.

81662 **Issue 7**81663 Austin Group Interpretation 1003.1-2001 #196 is applied, changing the SYNOPSIS to allow **-p** to
81664 be used with **-v** (or **-V**).

81665 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

81666 The *command* utility is moved from the User Portability Utilities option to the Base. User
81667 Portability Utilities is now an option for interactive utilities.81668 The APPLICATION USAGE and EXAMPLES are revised to replace the non-standard
81669 *getconf_CS_PATH* with *getconf PATH*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

compress

Utilities

81670 **NAME**81671 `compress` — compress data81672 **SYNOPSIS**81673 XSI `compress [-fv] [-b bits] [file...]`81674 `compress [-cfv] [-b bits] [file]`81675 **DESCRIPTION**81676 The *compress* utility shall attempt to reduce the size of the named files by using adaptive Lempel-
81677 Ziv coding algorithm.81678 **Note:** Lempel-Ziv is US Patent 4464650, issued to William Eastman, Abraham Lempel, Jacob Ziv,
81679 Martin Cohn on August 7th, 1984, and assigned to Sperry Corporation.81680 Lempel-Ziv-Welch compression is covered by US Patent 4558302, issued to Terry A. Welch on
81681 December 10th, 1985, and assigned to Sperry Corporation.81682 On systems not supporting adaptive Lempel-Ziv coding algorithm, the input files shall not be
81683 changed and an error value greater than two shall be returned. Except when the output is to the
81684 standard output, each file shall be replaced by one with the extension *.Z*. If the invoking process
81685 has appropriate privileges, the ownership, modes, access time, and modification time of the
81686 original file are preserved. If appending the *.Z* to the filename would make the name exceed
81687 {NAME_MAX} bytes, the command shall fail. If no files are specified, the standard input shall be
81688 compressed to the standard output.81689 **OPTIONS**81690 The *compress* utility shall conform to XBD Section 12.2 (on page 215).

81691 The following options shall be supported:

81692 **-b *bits*** Specify the maximum number of bits to use in a code. For a conforming
81693 application, the *bits* argument shall be:81694 $9 \leq bits \leq 14$ 81695 The implementation may allow *bits* values of greater than 14. The default is 14, 15,
81696 or 16.81697 **-c** Cause *compress* to write to the standard output; the input file is not changed, and
81698 no *.Z* files are created.81699 **-f** Force compression of *file*, even if it does not actually reduce the size of the file, or if
81700 the corresponding *file.Z* file already exists. If the **-f** option is not given, and the
81701 process is not running in the background, the user is prompted as to whether an
81702 existing *file.Z* file should be overwritten. If the response is affirmative, the existing
81703 file will be overwritten.81704 **-v** Write the percentage reduction of each file to standard error.81705 **OPERANDS**

81706 The following operand shall be supported:

81707 *file* A pathname of a file to be compressed.81708 **STDIN**81709 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is *'-'*.

81710 **INPUT FILES**

81711 If *file* operands are specified, the input files contain the data to be compressed.

81712 **ENVIRONMENT VARIABLES**

81713 The following environment variables shall affect the execution of *compress*:

81714 **LANG** Provide a default value for the internationalization variables that are unset or null.
 81715 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 81716 variables used to determine the values of locale categories.)

81717 **LC_ALL** If set to a non-empty string value, override the values of all the other
 81718 internationalization variables.

81719 **LC_COLLATE**

81720 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 81721 character collating elements used in the extended regular expression defined for
 81722 the **yesexpr** locale keyword in the **LC_MESSAGES** category.

81723 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 81724 characters (for example, single-byte as opposed to multi-byte characters in
 81725 arguments), the behavior of character classes used in the extended regular
 81726 expression defined for the **yesexpr** locale keyword in the **LC_MESSAGES** category.

81727 **LC_MESSAGES**

81728 Determine the locale used to process affirmative responses, and the locale used to
 81729 affect the format and contents of diagnostic messages, prompts, and the output
 81730 from the **-v** option written to standard error.

81731 **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

81732 **ASYNCHRONOUS EVENTS**

81733 Default.

81734 **STDOUT**

81735 If no *file* operands are specified, or if a *file* operand is **'-'**, or if the **-c** option is specified, the
 81736 standard output contains the compressed output.

81737 **STDERR**

81738 The standard error shall be used only for diagnostic and prompt messages and the output from
 81739 **-v**.

81740 **OUTPUT FILES**

81741 The output files shall contain the compressed output. The format of compressed files is
 81742 unspecified and interchange of such files between implementations (including access via
 81743 unspecified file sharing mechanisms) is not required by POSIX.1-2008.

81744 **EXTENDED DESCRIPTION**

81745 None.

81746 **EXIT STATUS**

81747 The following exit values shall be returned:

81748 0 Successful completion.

81749 1 An error occurred.

81750 2 One or more files were not compressed because they would have increased in size (and the
 81751 **-f** option was not specified).

compress

Utilities

81752 >2 An error occurred.

81753 **CONSEQUENCES OF ERRORS**

81754 The input file shall remain unmodified.

81755 **APPLICATION USAGE**

81756 The amount of compression obtained depends on the size of the input, the number of *bits* per
81757 code, and the distribution of common substrings. Typically, text such as source code or English is
81758 reduced by 50-60%. Compression is generally much better than that achieved by Huffman
81759 coding or adaptive Huffman coding (*compact*), and takes less time to compute.

81760 Although *compress* strictly follows the default actions upon receipt of a signal or when an error
81761 occurs, some unexpected results may occur. In some implementations it is likely that a partially
81762 compressed file is left in place, alongside its uncompressed input file. Since the general
81763 operation of *compress* is to delete the uncompressed file only after the *.Z* file has been
81764 successfully filled, an application should always carefully check the exit status of *compress* before
81765 arbitrarily deleting files that have like-named neighbors with *.Z* suffixes.

81766 The limit of 14 on the *bits* option-argument is to achieve portability to all systems (within the
81767 restrictions imposed by the lack of an explicit published file format). Some implementations
81768 based on 16-bit architectures cannot support 15 or 16-bit uncompression.

81769 **EXAMPLES**

81770 None.

81771 **RATIONALE**

81772 None.

81773 **FUTURE DIRECTIONS**

81774 None.

81775 **SEE ALSO**

81776 *uncompress*, *zcat*

81777 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

81778 **CHANGE HISTORY**

81779 First released in Issue 4.

81780 **Issue 6**

81781 The normative text is reworded to avoid use of the term “must” for application requirements.

81782 An error case is added for systems not supporting adaptive Lempel-Ziv coding.

81783 **Issue 7**

81784 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

81785 Austin Group Interpretation 1003.1-2001 #125 is applied, revising the ENVIRONMENT
81786 VARIABLES section.

Utilities

cp

NAME

cp — copy files

SYNOPSIS

cp [-Pfp] *source_file target_file*

cp [-Pfp] *source_file... target*

cp -R [-H|-L|-P] [-fip] *source_file... target*

DESCRIPTION

The first synopsis form is denoted by two operands, neither of which are existing files of type directory. The *cp* utility shall copy the contents of *source_file* (or, if *source_file* is a file of type symbolic link, the contents of the file referenced by *source_file*) to the destination path named by *target_file*.

The second synopsis form is denoted by two or more operands where the **-R** option is not specified and the first synopsis form is not applicable. It shall be an error if any *source_file* is a file of type directory, if *target* does not exist, or if *target* does not name a directory. The *cp* utility shall copy the contents of each *source_file* (or, if *source_file* is a file of type symbolic link, the contents of the file referenced by *source_file*) to the destination path named by the concatenation of *target*, a single <slash> character if *target* did not end in a <slash>, and the last component of *source_file*.

The third synopsis form is denoted by two or more operands where the **-R** option is specified. The *cp* utility shall copy each file in the file hierarchy rooted in each *source_file* to a destination path named as follows:

- If *target* exists and names an existing directory, the name of the corresponding destination path for each file in the file hierarchy shall be the concatenation of *target*, a single <slash> character if *target* did not end in a <slash>, and the pathname of the file relative to the directory containing *source_file*.
- If *target* does not exist and two operands are specified, the name of the corresponding destination path for *source_file* shall be *target*; the name of the corresponding destination path for all other files in the file hierarchy shall be the concatenation of *target*, a <slash> character, and the pathname of the file relative to *source_file*.

It shall be an error if *target* does not exist and more than two operands are specified, or if *target* exists and does not name a directory.

In the following description, the term *dest_file* refers to the file named by the destination path. The term *source_file* refers to the file that is being copied, whether specified as an operand or a file in a file hierarchy rooted in a *source_file* operand. If *source_file* is a file of type symbolic link:

- If the **-R** option was not specified, *cp* shall take actions based on the type and contents of the file referenced by the symbolic link, and not by the symbolic link itself, unless the **-P** option was specified.
- If the **-R** option was specified:
 - If none of the options **-H**, **-L**, nor **-P** were specified, it is unspecified which of **-H**, **-L**, or **-P** will be used as a default.
 - If the **-H** option was specified, *cp* shall take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand.
 - If the **-L** option was specified, *cp* shall take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand or any symbolic links encountered during traversal of a file hierarchy.

81831 — If the **-P** option was specified, *cp* shall copy any symbolic link specified as a
 81832 *source_file* operand and any symbolic links encountered during traversal of a file
 81833 hierarchy, and shall not follow any symbolic links.

81834 For each *source_file*, the following steps shall be taken:

- 81835 1. If *source_file* references the same file as *dest_file*, *cp* may write a diagnostic message to
 81836 standard error; it shall do nothing more with *source_file* and shall go on to any remaining
 81837 files.
- 81838 2. If *source_file* is of type directory, the following steps shall be taken:
 - 81839 a. If the **-R** option was not specified, *cp* shall write a diagnostic message to standard
 81840 error, do nothing more with *source_file*, and go on to any remaining files.
 - 81841 b. If *source_file* was not specified as an operand and *source_file* is dot or dot-dot, *cp*
 81842 shall do nothing more with *source_file* and go on to any remaining files.
 - 81843 c. If *dest_file* exists and it is a file type not specified by the System Interfaces volume
 81844 of POSIX.1-2008, the behavior is implementation-defined.
 - 81845 d. If *dest_file* exists and it is not of type directory, *cp* shall write a diagnostic message
 81846 to standard error, do nothing more with *source_file* or any files below *source_file* in
 81847 the file hierarchy, and go on to any remaining files.
 - 81848 e. If the directory *dest_file* does not exist, it shall be created with file permission bits
 81849 set to the same value as those of *source_file*, modified by the file creation mask of
 81850 the user if the **-p** option was not specified, and then bitwise-inclusively OR'ed
 81851 with S_IRWXU. If *dest_file* cannot be created, *cp* shall write a diagnostic message to
 81852 standard error, do nothing more with *source_file*, and go on to any remaining files.
 81853 It is unspecified if *cp* attempts to copy files in the file hierarchy rooted in *source_file*.
 - 81854 f. The files in the directory *source_file* shall be copied to the directory *dest_file*, taking
 81855 the four steps (1 to 4) listed here with the files as *source_files*.
 - 81856 g. If *dest_file* was created, its file permission bits shall be changed (if necessary) to be
 81857 the same as those of *source_file*, modified by the file creation mask of the user if the
 81858 **-p** option was not specified.
 - 81859 h. The *cp* utility shall do nothing more with *source_file* and go on to any remaining
 81860 files.
- 81861 3. If *source_file* is of type regular file, the following steps shall be taken:
 - 81862 a. The behavior is unspecified if *dest_file* exists and was written by a previous step.
 81863 Otherwise, if *dest_file* exists, the following steps shall be taken:
 - 81864 i. If the **-i** option is in effect, the *cp* utility shall write a prompt to the standard
 81865 error and read a line from the standard input. If the response is not
 81866 affirmative, *cp* shall do nothing more with *source_file* and go on to any
 81867 remaining files.
 - 81868 ii. A file descriptor for *dest_file* shall be obtained by performing actions
 81869 equivalent to the *open()* function defined in the System Interfaces volume of
 81870 POSIX.1-2008 called using *dest_file* as the *path* argument, and the bitwise-
 81871 inclusive OR of O_WRONLY and O_TRUNC as the *oflag* argument.
 - 81872 iii. If the attempt to obtain a file descriptor fails and the **-f** option is in effect, *cp*
 81873 shall attempt to remove the file by performing actions equivalent to the
 81874 *unlink()* function defined in the System Interfaces volume of POSIX.1-2008

- called using *dest_file* as the *path* argument. If this attempt succeeds, *cp* shall continue with step 3b.
- b. If *dest_file* does not exist, a file descriptor shall be obtained by performing actions equivalent to the *open()* function defined in the System Interfaces volume of POSIX.1-2008 called using *dest_file* as the *path* argument, and the bitwise-inclusive OR of *O_WRONLY* and *O_CREAT* as the *oflag* argument. The file permission bits of *source_file* shall be the *mode* argument.
 - c. If the attempt to obtain a file descriptor fails, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files.
 - d. The contents of *source_file* shall be written to the file descriptor. Any write errors shall cause *cp* to write a diagnostic message to standard error and continue to step 3e.
 - e. The file descriptor shall be closed.
 - f. The *cp* utility shall do nothing more with *source_file*. If a write error occurred in step 3d, it is unspecified if *cp* continues with any remaining files. If no write error occurred in step 3d, *cp* shall go on to any remaining files.
4. Otherwise, the **-R** option was specified, and the following steps shall be taken:
- a. The *dest_file* shall be created with the same file type as *source_file*.
 - b. If *source_file* is a file of type FIFO, the file permission bits shall be the same as those of *source_file*, modified by the file creation mask of the user if the **-p** option was not specified. Otherwise, the permissions, owner ID, and group ID of *dest_file* are implementation-defined.
- If this creation fails for any reason, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files.
- c. If *source_file* is a file of type symbolic link, and the options require the symbolic link itself to be acted upon, the pathname contained in *dest_file* shall be the same as the pathname contained in *source_file*.
- If this fails for any reason, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files.
- If the implementation provides additional or alternate access control mechanisms (see XBD Section 4.4, on page 108), their effect on copies of files is implementation-defined.

OPTIONS

The *cp* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

-f	If a file descriptor for a destination file cannot be obtained, as described in step 3.a.ii., attempt to unlink the destination file and proceed.
-H	Take actions based on the type and contents of the file referenced by any symbolic link specified as a <i>source_file</i> operand.
-i	Write a prompt to standard error before copying to any existing non-directory destination file. If the response from the standard input is affirmative, the copy shall be attempted; otherwise, it shall not.

cp*Utilities*

81916	-L	Take actions based on the type and contents of the file referenced by any symbolic link specified as a <i>source_file</i> operand or any symbolic links encountered during traversal of a file hierarchy.
81917		
81918		
81919	-P	Take actions on any symbolic link specified as a <i>source_file</i> operand or any symbolic link encountered during traversal of a file hierarchy.
81920		
81921	-p	Duplicate the following characteristics of each source file in the corresponding destination file:
81922		
81923		1. The time of last data modification and time of last access. If this duplication fails for any reason, <i>cp</i> shall write a diagnostic message to standard error.
81924		
81925		2. The user ID and group ID. If this duplication fails for any reason, it is unspecified whether <i>cp</i> writes a diagnostic message to standard error.
81926		
81927		3. The file permission bits and the S_ISUID and S_ISGID bits. Other, implementation-defined, bits may be duplicated as well. If this duplication fails for any reason, <i>cp</i> shall write a diagnostic message to standard error.
81928		
81929		
81930		If the user ID or the group ID cannot be duplicated, the file permission bits S_ISUID and S_ISGID shall be cleared. If these bits are present in the source file but are not duplicated in the destination file, it is unspecified whether <i>cp</i> writes a diagnostic message to standard error.
81931		
81932		
81933		
81934		The order in which the preceding characteristics are duplicated is unspecified. The <i>dest_file</i> shall not be deleted if these characteristics cannot be preserved.
81935		
81936		
81937	-R	Copy file hierarchies.
81938		Specifying more than one of the mutually-exclusive options -H, -L, and -P shall not be considered an error. The last option specified shall determine the behavior of the utility.
81939	OPERANDS	
81940		The following operands shall be supported:
81941	<i>source_file</i>	A pathname of a file to be copied. If a <i>source_file</i> operand is '-', it shall refer to a file named -; implementations shall not treat it as meaning standard input.
81942		
81943	<i>target_file</i>	A pathname of an existing or nonexistent file, used for the output when a single file is copied. If a <i>target_file</i> operand is '-', it shall refer to a file named -; implementations shall not treat it as meaning standard output.
81944		
81945		
81946	<i>target</i>	A pathname of a directory to contain the copied files.
81947	STDIN	
81948		The standard input shall be used to read an input line in response to each prompt specified in the STDERR section. Otherwise, the standard input shall not be used.
81949		
81950	INPUT FILES	
81951		The input files specified as operands may be of any file type.
81952	ENVIRONMENT VARIABLES	
81953		The following environment variables shall affect the execution of <i>cp</i> :
81954	LANG	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
81955		
81956		

81957	LC_ALL	If set to a non-empty string value, override the values of all the other internationalization variables.
81958		
81959	LC_COLLATE	Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the extended regular expression defined for the yesexpr locale keyword in the LC_MESSAGES category.
81960		
81961		
81962		
81963	LC_CTYPE	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes used in the extended regular expression defined for the yesexpr locale keyword in the LC_MESSAGES category.
81964		
81965		
81966		
81967		
81968	LC_MESSAGES	Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of diagnostic messages and prompts written to standard error.
81969		
81970		
81971		
81972	XSI NLSPATH	Determine the location of message catalogs for the processing of LC_MESSAGES .
81973	ASYNCHRONOUS EVENTS	
81974	Default.	
81975	STDOUT	
81976	Not used.	
81977	STDERR	
81978		A prompt shall be written to standard error under the conditions specified in the DESCRIPTION section. The prompt shall contain the destination pathname, but its format is otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.
81979		
81980		
81981	OUTPUT FILES	
81982	The output files may be of any type.	
81983	EXTENDED DESCRIPTION	
81984	None.	
81985	EXIT STATUS	
81986	The following exit values shall be returned:	
81987	0 All files were copied successfully.	
81988	>0 An error occurred.	
81989	CONSEQUENCES OF ERRORS	
81990	If cp is prematurely terminated by a signal or error, files or file hierarchies may be only partially copied and files and directories may have incorrect permissions or access and modification times.	
81991		
81992		

APPLICATION USAGE

The set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to prevent users from creating programs that are set-user-ID or set-group-ID to them when copying files or to make set-user-ID or set-group-ID files accessible to new groups of users. For example, if a file is set-user-ID and the copy has a different group ID than the source, a new group of users has execute permission to a set-user-ID program than did previously. In particular, this is a problem for superusers copying users' trees.

EXAMPLES

None.

RATIONALE

The `-i` option exists on BSD systems, giving applications and users a way to avoid accidentally removing files when copying. Although the 4.3 BSD version does not prompt if the standard input is not a terminal, the standard developers decided that use of `-i` is a request for interaction, so when the destination path exists, the utility takes instructions from whatever responds on standard input.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because implementations may desire more descriptive prompts than those used on historical implementations. Therefore, an application using the `-i` option relies on the system to provide the most suitable dialog directly with the user, based on the behavior specified.

The `-p` option is historical practice on BSD systems, duplicating the time of last data modification and time of last access. This volume of POSIX.1-2008 extends it to preserve the user and group IDs, as well as the file permissions. This requirement has obvious problems in that the directories are almost certainly modified after being copied. This volume of POSIX.1-2008 requires that the modification times be preserved. The statement that the order in which the characteristics are duplicated is unspecified is to permit implementations to provide the maximum amount of security for the user. Implementations should take into account the obvious security issues involved in setting the owner, group, and mode in the wrong order or creating files with an owner, group, or mode different from the final value.

It is unspecified whether `cp` writes diagnostic messages when the user and group IDs cannot be set due to the widespread practice of users using `-p` to duplicate some portion of the file characteristics, indifferent to the duplication of others. Historic implementations only write diagnostic messages on errors other than [EPERM].

Earlier versions of this standard included support for the `-r` option to copy file hierarchies. The `-r` option is historical practice on BSD and BSD-derived systems. This option is no longer specified by POSIX.1-2008 but may be present in some implementations. The `-R` option was added as a close synonym to the `-r` option, selected for consistency with all other options in this volume of POSIX.1-2008 that do recursive directory descent.

The difference between `-R` and the removed `-r` option is in the treatment by `cp` of file types other than regular and directory. It was implementation-defined how the `-` option treated special files to allow both historical implementations and those that chose to support `-r` with the same abilities as `-R` defined by this volume of POSIX.1-2008. The original `-r` flag, for historic reasons, did not handle special files any differently from regular files, but always read the file and copied its contents. This had obvious problems in the presence of special file types; for example, character devices, FIFOs, and sockets.

When a failure occurs during the copying of a file hierarchy, `cp` is required to attempt to copy files that are on the same level in the hierarchy or above the file where the failure occurred. It is unspecified if `cp` shall attempt to copy files below the file where the failure occurred (which

cannot succeed in any case).

Permissions, owners, and groups of created special file types have been deliberately left as implementation-defined. This is to allow systems to satisfy special requirements (for example, allowing users to create character special devices, but requiring them to be owned by a certain group). In general, it is strongly suggested that the permissions, owner, and group be the same as if the user had run the historical *mknod*, *ln*, or other utility to create the file. It is also probable that additional privileges are required to create block, character, or other implementation-defined special file types.

Additionally, the *-p* option explicitly requires that all set-user-ID and set-group-ID permissions be discarded if any of the owner or group IDs cannot be set. This is to keep users from unintentionally giving away special privilege when copying programs.

When creating regular files, historical versions of *cp* use the mode of the source file as modified by the file mode creation mask. Other choices would have been to use the mode of the source file unmodified by the creation mask or to use the same mode as would be given to a new file created by the user (plus the execution bits of the source file) and then modify it by the file mode creation mask. In the absence of any strong reason to change historic practice, it was in large part retained.

When creating directories, historical versions of *cp* use the mode of the source directory, plus read, write, and search bits for the owner, as modified by the file mode creation mask. This is done so that *cp* can copy trees where the user has read permission, but the owner does not. A side-effect is that if the file creation mask denies the owner permissions, *cp* fails. Also, once the copy is done, historical versions of *cp* set the permissions on the created directory to be the same as the source directory, unmodified by the file creation mask.

This behavior has been modified so that *cp* is always able to create the contents of the directory, regardless of the file creation mask. After the copy is done, the permissions are set to be the same as the source directory, as modified by the file creation mask. This latter change from historical behavior is to prevent users from accidentally creating directories with permissions beyond those they would normally set and for consistency with the behavior of *cp* in creating files.

It is not a requirement that *cp* detect attempts to copy a file to itself; however, implementations are strongly encouraged to do so. Historical implementations have detected the attempt in most cases.

There are two methods of copying subtrees in this volume of POSIX.1-2008. The other method is described as part of the *pax* utility (see *pax*). Both methods are historical practice. The *cp* utility provides a simpler, more intuitive interface, while *pax* offers a finer granularity of control. Each provides additional functionality to the other; in particular, *pax* maintains the hard-link structure of the hierarchy, while *cp* does not. It is the intention of the standard developers that the results be similar (using appropriate option combinations in both utilities). The results are not required to be identical; there seemed insufficient gain to applications to balance the difficulty of implementations having to guarantee that the results would be exactly identical.

The wording allowing *cp* to copy a directory to implementation-defined file types not specified by the System Interfaces volume of POSIX.1-2008 is provided so that implementations supporting symbolic links are not required to prohibit copying directories to symbolic links. Other extensions to the System Interfaces volume of POSIX.1-2008 file types may need to use this loophole as well.

82085 **FUTURE DIRECTIONS**

82086 None.

82087 **SEE ALSO**82088 *mv, find, ln, pax*82089 XBD [Section 4.4](#) (on page 108), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)82090 XSH *open()*, *unlink()*82091 **CHANGE HISTORY**

82092 First released in Issue 2.

82093 **Issue 6**82094 The **-r** option is marked obsolescent.82095 The new options **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These
82096 options affect the processing of symbolic links.82097 IEEE PASC Interpretation 1003.2 #194 is applied, adding a description of the **-P** option.82098 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/18 is applied, correcting an error in the
82099 SEE ALSO section.82100 **Issue 7**82101 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the
82102 *LC_MESSAGES* environment variable.

82103 Austin Group Interpretations 1003.1-2001 #092, #164, #165, and #168 are applied.

82104 SD5-XCU-ERN-31 and SD5-XCU-ERN-42 are applied, updating the DESCRIPTION.

82105 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82106 SD5-XCU-ERN-102 is applied, clarifying the **-i** option within the OPTIONS section.82107 The obsolescent **-r** option is removed.82108 The **-P** option is added to the SYNOPSIS and to the DESCRIPTION with respect to the **-R**
82109 option.

Utilities

crontab82110 **NAME**82111 `crontab` — schedule periodic background work82112 **SYNOPSIS**82113 `crontab` [*file*]82114 UP `crontab` [**-e** | **-l** | **-r**]82115 **DESCRIPTION**

82116 UP The *crontab* utility shall create, replace, or edit a user's crontab entry; a crontab entry is a list of
 82117 commands and the times at which they shall be executed. The new crontab entry can be input by
 82118 UP specifying *file* or input from standard input if no *file* operand is specified, or by using an editor,
 82119 if **-e** is specified.

82120 Upon execution of a command from a crontab entry, the implementation shall supply a default
 82121 environment, defining at least the following environment variables:

82122 *HOME* A pathname of the user's home directory.

82123 *LOGNAME* The user's login name.

82124 *PATH* A string representing a search path guaranteed to find all of the standard utilities.

82125 *SHELL* A pathname of the command interpreter. When *crontab* is invoked as specified by
 82126 this volume of POSIX.1-2008, the value shall be a pathname for *sh*.

82127 The values of these variables when *crontab* is invoked as specified by this volume of
 82128 POSIX.1-2008 shall not affect the default values provided when the scheduled command is run.

82129 If standard output and standard error are not redirected by commands executed from the
 82130 crontab entry, any generated output or errors shall be mailed, via an implementation-defined
 82131 method, to the user.

82132 XSI Users shall be permitted to use *crontab* if their names appear in the file **cron.allow** which is
 82133 located in an implementation-defined directory. If that file does not exist, the file **cron.deny**,
 82134 which is located in an implementation-defined directory, shall be checked to determine whether
 82135 the user shall be denied access to *crontab*. If neither file exists, only a process with appropriate
 82136 privileges shall be allowed to submit a job. If only **cron.deny** exists and is empty, global usage
 82137 shall be permitted. The **cron.allow** and **cron.deny** files shall consist of one user name per line.

82138 **OPTIONS**

82139 The *crontab* utility shall conform to XBD Section 12.2 (on page 215).

82140 The following options shall be supported:

82141 UP **-e** Edit a copy of the invoking user's crontab entry, or create an empty entry to edit if
 82142 the crontab entry does not exist. When editing is complete, the entry shall be
 82143 installed as the user's crontab entry.

82144 **-l** (The letter ell.) List the invoking user's crontab entry.

82145 **-r** Remove the invoking user's crontab entry.

82146 **OPERANDS**

82147 The following operand shall be supported:

82148 *file* The pathname of a file that contains specifications, in the format defined in the
 82149 INPUT FILES section, for crontab entries.

crontab*Utilities*82150 **STDIN**

82151 See the INPUT FILES section.

82152 **INPUT FILES**

82153 In the POSIX locale, the user or application shall ensure that a crontab entry is a text file
 82154 consisting of lines of six fields each. The fields shall be separated by <blank> characters. The
 82155 first five fields shall be integer patterns that specify the following:

- 82156 1. Minute [0,59]
- 82157 2. Hour [0,23]
- 82158 3. Day of the month [1,31]
- 82159 4. Month of the year [1,12]
- 82160 5. Day of the week ([0,6] with 0=Sunday)

82161 Each of these patterns can be either an <asterisk> (meaning all valid values), an element, or a list
 82162 of elements separated by <comma> characters. An element shall be either a number or two
 82163 numbers separated by a <hyphen> (meaning an inclusive range). The specification of days can
 82164 be made by two fields (day of the month and day of the week). If month, day of month, and day
 82165 of week are all <asterisk> characters, every day shall be matched. If either the month or day of
 82166 month is specified as an element or list, but the day of week is an <asterisk>, the month and day
 82167 of month fields shall specify the days that match. If both month and day of month are specified
 82168 as an <asterisk>, but day of week is an element or list, then only the specified days of the week
 82169 match. Finally, if either the month or day of month is specified as an element or list, and the day
 82170 of week is also specified as an element or list, then any day matching either the month and day
 82171 of month, or the day of week, shall be matched.

82172 The sixth field of a line in a crontab entry is a string that shall be executed by *sh* at the specified
 82173 times. A <percent-sign> character in this field shall be translated to a <newline>. Any character
 82174 preceded by a <backslash> (including the ' % ') shall cause that character to be treated literally.
 82175 Only the first line (up to a ' % ' or end-of-line) of the command field shall be executed by the
 82176 command interpreter. The other lines shall be made available to the command as standard input.

82177 Blank lines and those whose first non-<blank> is ' # ' shall be ignored.

82178 XSI The text files **cron.allow** and **cron.deny**, which are located in an implementation-defined
 82179 directory, shall contain zero or more user names, one per line, of users who are, respectively,
 82180 authorized or denied access to the service underlying the *crontab* utility.

82181 **ENVIRONMENT VARIABLES**

82182 The following environment variables shall affect the execution of *crontab*:

82183	<i>EDITOR</i>	Determine the editor to be invoked when the -e option is specified. The default editor shall be <i>vi</i> .
82184		
82185	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
82186		
82187		
82188	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
82189		
82190	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).
82191		
82192		

82193	<i>LC_MESSAGES</i>	
82194		Determine the locale that should be used to affect the format and contents of
82195		diagnostic messages written to standard error.
82196	XSI	<i>NLSPATH</i> Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
82197	ASYNCHRONOUS EVENTS	
82198		Default.
82199	STDOUT	
82200		If the <i>-l</i> option is specified, the crontab entry shall be written to the standard output.
82201	STDERR	
82202		The standard error shall be used only for diagnostic messages.
82203	OUTPUT FILES	
82204		None.
82205	EXTENDED DESCRIPTION	
82206		None.
82207	EXIT STATUS	
82208		The following exit values shall be returned:
82209	0	Successful completion.
82210	>0	An error occurred.
82211	CONSEQUENCES OF ERRORS	
82212	UP	The user's crontab entry is not submitted, removed, edited , or listed.
82213	APPLICATION USAGE	
82214		The format of the crontab entry shown here is guaranteed only for the POSIX locale. Other
82215		cultures may be supported with substantially different interfaces, although implementations are
82216		encouraged to provide comparable levels of functionality.
82217		The default settings of the <i>HOME</i> , <i>LOGNAME</i> , <i>PATH</i> , and <i>SHELL</i> variables that are given to the
82218		scheduled job are not affected by the settings of those variables when <i>crontab</i> is run; as stated,
82219		they are defaults. The text about "invoked as specified by this volume of POSIX.1-2008" means
82220		that the implementation may provide extensions that allow these variables to be affected at
82221		runtime, but that the user has to take explicit action in order to access the extension, such as give
82222		a new option flag or modify the format of the crontab entry.
82223		A typical user error is to type only <i>crontab</i> ; this causes the system to wait for the new crontab
82224		entry on standard input. If end-of-file is typed (generally <control>-D), the crontab entry is
82225		replaced by an empty file. In this case, the user should type the interrupt character, which
82226		prevents the crontab entry from being replaced.
82227	EXAMPLES	
82228	1.	Clean up core files every weekday morning at 3:15 am:
82229		15 3 * * 1-5 find "\$HOME" -name core -exec rm -f {} + 2>/dev/null
82230	2.	Mail a birthday greeting:
82231		0 12 14 2 * mailx john%Happy Birthday!%Time for lunch.
82232	3.	As an example of specifying the two types of days:
82233		0 0 1,15 * 1

crontab*Utilities*

82234 would run a command on the first and fifteenth of each month, as well as on every
 82235 Monday. To specify days by only one field, the other field should be set to `'*'`; for
 82236 example:

82237 `0 0 * * 1`

82238 would run a command only on Mondays.

RATIONALE

82240 All references to a *cron* daemon and to *cron files* have been omitted. Although historical
 82241 implementations have used this arrangement, there is no reason to limit future implementations.

82242 This description of *crontab* is designed to support only users with normal privileges. The format
 82243 of the input is based on the System V *crontab*; however, there is no requirement here that the
 82244 actual system database used by the *cron* daemon (or a similar mechanism) use this format
 82245 internally. For example, systems derived from BSD are likely to have an additional field
 82246 appended that indicates the user identity to be used when the job is submitted.

82247 The `-e` option was adopted from the SVID as a user convenience, although it does not exist in all
 82248 historical implementations.

FUTURE DIRECTIONS

82249 None.

SEE ALSO

82252 *at*

82253 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

CHANGE HISTORY

82255 First released in Issue 2.

Issue 6

82257 This utility is marked as part of the User Portability Utilities option.

82258 The normative text is reworded to avoid use of the term “must” for application requirements.

Issue 7

82260 The *crontab* utility (except for the `-e` option) is moved from the User Portability Utilities option
 82261 to the Base. User Portability Utilities is now an option for interactive utilities.

82262 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced
 82263 by the *crontab* utility.

82264 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82265 The first example is changed to remove the unreliable use of `find | xargs`.

Utilities

csplit

NAME

csplit — split files based on context

SYNOPSIS

csplit [-ks] [-f *prefix*] [-n *number*] *file arg...*

DESCRIPTION

The *csplit* utility shall read the file named by the *file* operand, write all or part of that file into other files as directed by the *arg* operands, and write the sizes of the files.

OPTIONS

The *csplit* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

- f *prefix*** Name the created files *prefix00*, *prefix01*, ..., *prefixn*. The default is *xx00* ... *xxn*. If the *prefix* argument would create a filename exceeding {NAME_MAX} bytes, an error shall result, *csplit* shall exit with a diagnostic message, and no files shall be created.
- k** Leave previously created files intact. By default, *csplit* shall remove created files if an error occurs.
- n *number*** Use *number* decimal digits to form filenames for the file pieces. The default shall be 2.
- s** Suppress the output of file size messages.

OPERANDS

The following operands shall be supported:

file The pathname of a text file to be split. If *file* is '-', the standard input shall be used.

Each *arg* operand can be one of the following:

/rexp/[offset]

A file shall be created using the content of the lines from the current line up to, but not including, the line that results from the evaluation of the regular expression with *offset*, if any, applied. The regular expression *rexp* shall follow the rules for basic regular expressions described in XBD Section 9.3 (on page 183). The application shall use the sequence "\/" to specify a <slash> character within the *rexp*. The optional offset shall be a positive or negative integer value representing a number of lines. A positive integer value can be preceded by '+'. If the selection of lines from an *offset* expression of this type would create a file with zero lines, or one with greater than the number of lines left in the input file, the results are unspecified. After the section is created, the current line shall be set to the line that results from the evaluation of the regular expression with any offset applied. If the current line is the first line in the file and a regular expression operation has not yet been performed, the pattern match of *rexp* shall be applied from the current line to the end of the file. Otherwise, the pattern match of *rexp* shall be applied from the line following the current line to the end of the file.

%rexp%[offset]

Equivalent to */rexp/[offset]*, except that no file shall be created for the selected section of the input file. The application shall use the sequence "%" to specify a <percent-sign> character within the *rexp*.

csplit

Utilities

- 82310 *line_no* Create a file from the current line up to (but not including) the line number *line_no*.
 82311 Lines in the file shall be numbered starting at one. The current line becomes
 82312 *line_no*.
- 82313 {*num*} Repeat operand. This operand can follow any of the operands described
 82314 previously. If it follows a *rexp* type operand, that operand shall be applied *num*
 82315 more times. If it follows a *line_no* operand, the file shall be split every *line_no* lines,
 82316 *num* times, from that point.
- 82317 An error shall be reported if an operand does not reference a line between the current position
 82318 and the end of the file.
- 82319 **STDIN**
- 82320 See the INPUT FILES section.
- 82321 **INPUT FILES**
- 82322 The input file shall be a text file.
- 82323 **ENVIRONMENT VARIABLES**
- 82324 The following environment variables shall affect the execution of *csplit*:
- 82325 *LANG* Provide a default value for the internationalization variables that are unset or null.
 82326 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 82327 variables used to determine the values of locale categories.)
- 82328 *LC_ALL* If set to a non-empty string value, override the values of all the other
 82329 internationalization variables.
- 82330 *LC_COLLATE*
 82331 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 82332 character collating elements within regular expressions.
- 82333 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 82334 characters (for example, single-byte as opposed to multi-byte characters in
 82335 arguments and input files) and the behavior of character classes within regular
 82336 expressions.
- 82337 *LC_MESSAGES*
 82338 Determine the locale that should be used to affect the format and contents of
 82339 diagnostic messages written to standard error.
- 82340 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 82341 **ASYNCHRONOUS EVENTS**
- 82342 If the **-k** option is specified, created files shall be retained. Otherwise, the default action occurs.
- 82343 **STDOUT**
- 82344 Unless the **-s** option is used, the standard output shall consist of one line per file created, with a
 82345 format as follows:
- 82346 "%d\n", <file size in bytes>
- 82347 **STDERR**
- 82348 The standard error shall be used only for diagnostic messages.
- 82349 **OUTPUT FILES**
- 82350 The output files shall contain portions of the original input file; otherwise, unchanged.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

By default, created files shall be removed if an error occurs. When the **-k** option is specified, created files shall not be removed if an error occurs.

APPLICATION USAGE

None.

EXAMPLES

1. This example creates four files, **cobol00** ... **cobol03**:

```
csplit -f cobol file '/procedure division/' /par5./ /par16./
```

After editing the split files, they can be recombined as follows:

```
cat cobol0[0-3] > file
```

Note that this example overwrites the original file.

2. This example would split the file after the first 99 lines, and every 100 lines thereafter, up to 9 999 lines; this is because lines in the file are numbered from 1 rather than zero, for historical reasons:

```
csplit -k file 100 {99}
```

3. Assuming that **prog.c** follows the C-language coding convention of ending routines with a **'}'** at the beginning of the line, this example creates a file containing each separate C routine (up to 21) in **prog.c**:

```
csplit -k prog.c '%main(%' '/^}/+1' {20}
```

RATIONALE

The **-n** option was added to extend the range of filenames that could be handled.

Consideration was given to adding a **-a** flag to use the alphabetic filename generation used by the historical *split* utility, but the functionality added by the **-n** option was deemed to make alphabetic naming unnecessary.

FUTURE DIRECTIONS

None.

SEE ALSO

sed, *split*

XBD [Chapter 8](#) (on page 173), [Section 9.3](#) (on page 183), [Section 12.2](#) (on page 215)

CHANGE HISTORY

First released in Issue 2.

csplit*Utilities*82388 **Issue 5**

82389 The FUTURE DIRECTIONS section is added.

82390 **Issue 6**

82391 This utility is marked as part of the User Portability Utilities option.

82392 The APPLICATION USAGE section is added.

82393 The description of regular expression operands is changed to align with the IEEE P1003.2b draft
82394 standard.

82395 The normative text is reworded to avoid use of the term “must” for application requirements.

82396 **Issue 7**

82397 The *csplit* utility is moved from the User Portability Utilities option to the Base User Portability
82398 Utilities is now an option for interactive utilities.

82399 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82400 The SYNOPSIS and OPERANDS sections are revised to use a single *arg* to split a file into two
82401 pieces.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

ctags

NAME

ctags — create a tags file (DEVELOPMENT, FORTRAN)

SYNOPSIS

```
SD      ctags [-a] [-f tagsfile] pathname...
82406   ctags -x pathname...
```

DESCRIPTION

The *ctags* utility shall be provided on systems that support the the Software Development Utilities option, and either or both of the C-Language Development Utilities option and FORTRAN Development Utilities option. On other systems, it is optional.

The *ctags* utility shall write a *tagsfile* or an index of objects from C-language or FORTRAN source files specified by the *pathname* operands. The *tagsfile* shall list the locators of language-specific objects within the source files. A locator consists of a name, *pathname*, and either a search pattern or a line number that can be used in searching for the object definition. The objects that shall be recognized are specified in the EXTENDED DESCRIPTION section.

OPTIONS

The *ctags* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

- a** Append to *tagsfile*.
- f *tagsfile*** Write the object locator lists into *tagsfile* instead of the default file named **tags** in the current directory.
- x** Produce a list of object names, the line number, and filename in which each is defined, as well as the text of that line, and write this to the standard output. A *tagsfile* shall not be created when **-x** is specified.

OPERANDS

The following *pathname* operands are supported:

- file.c*** Files with basenames ending with the **.c** suffix shall be treated as C-language source code. Such files that are not valid input to *c99* produce unspecified results.
- file.h*** Files with basenames ending with the **.h** suffix shall be treated as C-language source code. Such files that are not valid input to *c99* produce unspecified results.
- file.f*** Files with basenames ending with the **.f** suffix shall be treated as FORTRAN-language source code. Such files that are not valid input to *fort77* produce unspecified results.

The handling of other files is implementation-defined.

STDIN

See the INPUT FILES section.

INPUT FILES

The input files shall be text files containing source code in the language indicated by the operand filename suffixes.

ENVIRONMENT VARIABLES

ctags*Utilities*

- 82441 The following environment variables shall affect the execution of *ctags*:
- 82442 *LANG* Provide a default value for the internationalization variables that are unset or null.
 82443 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 82444 variables used to determine the values of locale categories.)
- 82445 *LC_ALL* If set to a non-empty string value, override the values of all the other
 82446 internationalization variables.
- 82447 *LC_COLLATE*
 82448 Determine the order in which output is sorted for the *-x* option. The POSIX locale
 82449 determines the order in which the *tagsfile* is written.
- 82450 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 82451 characters (for example, single-byte as opposed to multi-byte characters in
 82452 arguments and input files). When processing C-language source code, if the locale
 82453 is not compatible with the C locale described by the ISO C standard, the results are
 82454 unspecified.
- 82455 *LC_MESSAGES*
 82456 Determine the locale that should be used to affect the format and contents of
 82457 diagnostic messages written to standard error.
- 82458 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 82459 **ASYNCHRONOUS EVENTS**
- 82460 Default.
- 82461 **STDOUT**
 82462 The list of object name information produced by the *-x* option shall be written to standard
 82463 output in the following format:
- 82464 "%s %d %s %s", <object-name>, <line-number>, <filename>, <text>
 82465 where <text> is the text of line <line-number> of file <filename>.
- 82466 **STDERR**
 82467 The standard error shall be used only for diagnostic messages.
- 82468 **OUTPUT FILES**
 82469 When the *-x* option is not specified, the format of the output file shall be:
- 82470 "%s\t%s\t%s\t%s\n", <identifier>, <filename>, <pattern>
 82471 where <pattern> is a search pattern that could be used by an editor to find the defining instance
 82472 of <identifier> in <filename> (where *defining instance* is indicated by the declarations listed in the
 82473 EXTENDED DESCRIPTION).
- 82474 An optional <circumflex> ('^') can be added as a prefix to <pattern>, and an optional <dollar-
 82475 sign> can be appended to <pattern> to indicate that the pattern is anchored to the beginning
 82476 (end) of a line of text. Any <slash> or <backslash> characters in <pattern> shall be preceded by a
 82477 <backslash> character. The anchoring <circumflex>, <dollar-sign>, and escaping <backslash>
 82478 characters shall not be considered part of the search pattern. All other characters in the search
 82479 pattern shall be considered literal characters.

An alternative format is:

```
"%s\t%s\t%s?\n", <identifier>, <filename>, <pattern>
```

which is identical to the first format except that <slash> characters in <pattern> shall not be preceded by escaping <backslash> characters, and <question-mark> characters in <pattern> shall be preceded by <backslash> characters.

A second alternative format is:

```
"%s\t%s\t%d\n", <identifier>, <filename>, <lineno>
```

where <lineno> is a decimal line number that could be used by an editor to find <identifier> in <filename>.

Neither alternative format shall be produced by *ctags* when it is used as described by POSIX.1-2008, but the standard utilities that process tags files shall be able to process those formats as well as the first format.

In any of these formats, the file shall be sorted by identifier, based on the collation sequence in the POSIX locale.

EXTENDED DESCRIPTION

If the operand identifies C-language source, the *ctags* utility shall attempt to produce an output line for each of the following objects:

- Function definitions
- Type definitions
- Macros with arguments

It may also produce output for any of the following objects:

- Function prototypes
- Structures
- Unions
- Global variable definitions
- Enumeration types
- Macros without arguments
- **#define** statements
- **#line** statements

Any **#if** and **#ifdef** statements shall produce no output. The tag **main** is treated specially in C programs. The tag formed shall be created by prefixing **M** to the name of the file, with the trailing **.c**, and leading pathname components (if any) removed.

On systems that do not support the C-Language Development Utilities option, *ctags* produces unspecified results for C-language source code files. It should write to standard error a message identifying this condition and cause a non-zero exit status to be produced.

If the operand identifies FORTRAN source, the *ctags* utility shall produce an output line for each function definition. It may also produce output for any of the following objects:

- Subroutine definitions

ctags*Utilities*

- 82518 • COMMON statements
- 82519 • PARAMETER statements
- 82520 • DATA and BLOCK DATA statements
- 82521 • Statement numbers

82522 On systems that do not support the FORTRAN Development Utilities option, *ctags* produces
 82523 unspecified results for FORTRAN source code files. It should write to standard error a message
 82524 identifying this condition and cause a non-zero exit status to be produced.

82525 It is implementation-defined what other objects (including duplicate identifiers) produce output.

EXIT STATUS

82526 The following exit values shall be returned:

- 82528 0 Successful completion.
- 82529 >0 An error occurred.

CONSEQUENCES OF ERRORS

82530 Default.

APPLICATION USAGE

82533 The output with *-x* is meant to be a simple index that can be written out as an off-line readable
 82534 function index. If the input files to *ctags* (such as *.c* files) were not created using the same locale
 82535 as that in effect when *ctags -x* is run, results might not be as expected.

82536 The description of C-language processing says “attempts to” because the C language can be
 82537 greatly confused, especially through the use of *#defines*, and this utility would be of no use if
 82538 the real C preprocessor were run to identify them. The output from *ctags* may be fooled and
 82539 incorrect for various constructs.

EXAMPLES

82540 None.

RATIONALE

82543 The option list was significantly reduced from that provided by historical implementations. The
 82544 *-F* option was omitted as redundant, since it is the default. The *-B* option was omitted as being
 82545 of very limited usefulness. The *-t* option was omitted since the recognition of *typedefs* is now
 82546 required for C source files. The *-u* option was omitted because the update function was judged
 82547 to be not only inefficient, but also rarely needed.

82548 An early proposal included a *-w* option to suppress warning diagnostics. Since the types of such
 82549 diagnostics could not be described, the option was omitted as being not useful.

82550 The text for *LC_CTYPE* about compatibility with the C locale acknowledges that the ISO C
 82551 standard imposes requirements on the locale used to process C source. This could easily be a
 82552 superset of that known as “the C locale” by way of implementation extensions, or one of a few
 82553 alternative locales for systems supporting different codesets. No statement is made for
 82554 FORTRAN because the ANSI X3.9-1978 standard (FORTRAN 77) does not (yet) define a similar
 82555 locale concept. However, a general rule in this volume of POSIX.1-2008 is that any time that
 82556 locales do not match (preparing a file for one locale and processing it in another), the results are
 82557 suspect.

82558 The collation sequence of the tags file is not affected by *LC_COLLATE* because it is typically not
 82559 used by human readers, but only by programs such as *vi* to locate the tag within the source files.
 82560 Using the POSIX locale eliminates some of the problems of coordinating locales between the
 82561 *ctags* file creator and the *vi* file reader.

Historically, the tags file has been used only by *ex* and *vi*. However, the format of the tags file has been published to encourage other programs to use the tags in new ways. The format allows either patterns or line numbers to find the identifiers because the historical *vi* recognizes either. The *ctags* utility does not produce the format using line numbers because it is not useful following any source file changes that add or delete lines. The documented search patterns match historical practice. It should be noted that literal leading <circumflex> or trailing <dollar-sign> characters in the search pattern will only behave correctly if anchored to the beginning of the line or end of the line by an additional <circumflex> or <dollar-sign> character.

Historical implementations also understand the objects used by the languages Pascal and sometimes LISP, and they understand the C source output by *lex* and *yacc*. The *ctags* utility is not required to accommodate these languages, although implementors are encouraged to do so.

The following historical option was not specified, as *vgrind* is not included in this volume of POSIX.1-2008:

-v If the **-v** flag is given, an index of the form expected by *vgrind* is produced on the standard output. This listing contains the function name, filename, and page number (assuming 64-line pages). Since the output is sorted into lexicographic order, it may be desired to run the output through *sort -f*. Sample use:

```
ctags -v files | sort -f > index.vgrind -x index
```

The special treatment of the tag **main** makes the use of *ctags* practical in directories with more than one program.

FUTURE DIRECTIONS

None.

SEE ALSO

c99, *fort77*, *vi*

XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The OUTPUT FILES section is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term “must” for application requirements.

IEEE PASC Interpretation 1003.2 #168 is applied, changing “create” to “write” in the DESCRIPTION.

Issue 7

The *ctags* utility is no longer dependent on support for the User Portability Utilities option.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

cut*Utilities*82600 **NAME**82601 **cut** — cut out selected fields of each line of a file82602 **SYNOPSIS**82603 `cut -b list [-n] [file...]`82604 `cut -c list [file...]`82605 `cut -f list [-d delim] [-s] [file...]`82606 **DESCRIPTION**

82607 The *cut* utility shall cut out bytes (**-b** option), characters (**-c** option), or character-delimited fields
 82608 (**-f** option) from each line in one or more files, concatenate them, and write them to standard
 82609 output.

82610 **OPTIONS**82611 The *cut* utility shall conform to XBD [Section 12.2](#) (on page 215).

82612 The application shall ensure that the option-argument *list* (see options **-b**, **-c**, and **-f** below) is a
 82613 <comma>-separated list or <blank>-separated list of positive numbers and ranges. Ranges can
 82614 be in three forms. The first is two positive numbers separated by a <hyphen> (*low-high*), which
 82615 represents all fields from the first number to the second number. The second is a positive
 82616 number preceded by a <hyphen> (*-high*), which represents all fields from field number 1 to that
 82617 number. The third is a positive number followed by a <hyphen> (*low-*), which represents that
 82618 number to the last field, inclusive. The elements in *list* can be repeated, can overlap, and can be
 82619 specified in any order, but the bytes, characters, or fields selected shall be written in the order of
 82620 the input data. If an element appears in the selection list more than once, it shall be written
 82621 exactly once.

82622 The following options shall be supported:

82623 **-b *list*** Cut based on a *list* of bytes. Each selected byte shall be output unless the **-n** option
 82624 is also specified. It shall not be an error to select bytes not present in the input line.

82625 **-c *list*** Cut based on a *list* of characters. Each selected character shall be output. It shall
 82626 not be an error to select characters not present in the input line.

82627 **-d *delim*** Set the field delimiter to the character *delim*. The default is the <tab>.

82628 **-f *list*** Cut based on a *list* of fields, assumed to be separated in the file by a delimiter
 82629 character (see **-d**). Each selected field shall be output. Output fields shall be
 82630 separated by a single occurrence of the field delimiter character. Lines with no field
 82631 delimiters shall be passed through intact, unless **-s** is specified. It shall not be an
 82632 error to select fields not present in the input line.

82633 **-n** Do not split characters. When specified with the **-b** option, each element in *list* of
 82634 the form *low-high* (<hyphen>-separated numbers) shall be modified as follows:

- If the byte selected by *low* is not the first byte of a character, *low* shall be decremented to select the first byte of the character originally selected by *low*. If the byte selected by *high* is not the last byte of a character, *high* shall be decremented to select the last byte of the character prior to the character originally selected by *high*, or zero if there is no prior character. If the resulting range element has *high* equal to zero or *low* greater than *high*, the list element shall be dropped from *list* for that input line without causing an error.

82643 Each element in *list* of the form *low-* shall be treated as above with *high* set to the
 82644 number of bytes in the current line, not including the terminating <newline>. Each

Utilities

cut

82645 element in *list* of the form *-high* shall be treated as above with *low* set to 1. Each
 82646 element in *list* of the form *num* (a single number) shall be treated as above with *low*
 82647 set to *num* and *high* set to *num*.

82648 **-s** Suppress lines with no delimiter characters, when used with the **-f** option. Unless
 82649 specified, lines with no delimiters shall be passed through untouched.

82650 **OPERANDS**

82651 The following operand shall be supported:

82652 *file* A pathname of an input file. If no *file* operands are specified, or if a *file* operand is
 82653 '*-*', the standard input shall be used.

82654 **STDIN**

82655 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '*-*'.
 82656 See the INPUT FILES section.

82657 **INPUT FILES**

82658 The input files shall be text files, except that line lengths shall be unlimited.

82659 **ENVIRONMENT VARIABLES**

82660 The following environment variables shall affect the execution of *cut*:

82661 *LANG* Provide a default value for the internationalization variables that are unset or null.
 82662 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 82663 variables used to determine the values of locale categories.)

82664 *LC_ALL* If set to a non-empty string value, override the values of all the other
 82665 internationalization variables.

82666 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 82667 characters (for example, single-byte as opposed to multi-byte characters in
 82668 arguments and input files).

82669 *LC_MESSAGES*

82670 Determine the locale that should be used to affect the format and contents of
 82671 diagnostic messages written to standard error.

82672 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

82673 **ASYNCHRONOUS EVENTS**

82674 Default.

82675 **STDOUT**

82676 The *cut* utility output shall be a concatenation of the selected bytes, characters, or fields (one of
 82677 the following):

82678 "%s\n", <concatenation of bytes>

82679 "%s\n", <concatenation of characters>

82680 "%s\n", <concatenation of fields and field delimiters>

82681 **STDERR**

82682 The standard error shall be used only for diagnostic messages.

82683 **OUTPUT FILES**

82684 None.

82685 EXTENDED DESCRIPTION

82686 None.

82687 EXIT STATUS

82688 The following exit values shall be returned:

82689 0 All input files were output successfully.

82690 >0 An error occurred.

82691 CONSEQUENCES OF ERRORS

82692 Default.

82693 APPLICATION USAGE

82694 The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths.
 82695 The *cut* utility should be used when the number of lines (or records) needs to remain constant.
 82696 The *fold* utility should be used when the contents of long lines need to be kept contiguous.

82697 Earlier versions of the *cut* utility worked in an environment where bytes and characters were
 82698 considered equivalent (modulo <backspace> and <tab> processing in some implementations). In
 82699 the extended world of multi-byte characters, the new **-b** option has been added. The **-n** option
 82700 (used with **-b**) allows it to be used to act on bytes rounded to character boundaries. The
 82701 algorithm specified for **-n** guarantees that:

82702 `cut -b 1-500 -n file > file1`82703 `cut -b 501- -n file > file2`

82704 ends up with all the characters in **file** appearing exactly once in **file1** or **file2**. (There is,
 82705 however, a <newline> in both **file1** and **file2** for each <newline> in **file**.)

82706 EXAMPLES

82707 Examples of the option qualifier list:

82708 1,4,7 Select the first, fourth, and seventh bytes, characters, or fields and field delimiters.

82709 1-3,8 Equivalent to 1,2,3,8.

82710 -5,10 Equivalent to 1,2,3,4,5,10.

82711 3- Equivalent to third to last, inclusive.

82712 The *low-high* forms are not always equivalent when used with **-b** and **-n** and multi-byte
 82713 characters; see the description of **-n**.

82714 The following command:

82715 `cut -d : -f 1,6 /etc/passwd`

82716 reads the System V password file (user database) and produces lines of the form:

82717 `<user ID>:<home directory>`

82718 Most utilities in this volume of POSIX.1-2008 work on text files. The *cut* utility can be used to
 82719 turn files with arbitrary line lengths into a set of text files containing the same data. The *paste*
 82720 utility can be used to create (or recreate) files with arbitrary line lengths. For example, if **file**
 82721 contains long lines:

82722 `cut -b 1-500 -n file > file1`82723 `cut -b 501- -n file > file2`

82724 creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that
 82725 contains the remainder of the data from **file**. (Note that **file2** is not a text file if there are lines in

file that are longer than 500 + {LINE_MAX} bytes.) The original file can be recreated from **file1** and **file2** using the command:

```
paste -d "\0" file1 file2 > file
```

RATIONALE

Some historical implementations do not count <backspace> characters in determining character counts with the **-c** option. This may be useful for using *cut* for processing *nroff* output. It was deliberately decided not to have the **-c** option treat either <backspace> or <tab> characters in any special fashion. The *fold* utility does treat these characters specially.

Unlike other utilities, some historical implementations of *cut* exit after not finding an input file, rather than continuing to process the remaining *file* operands. This behavior is prohibited by this volume of POSIX.1-2008, where only the exit status is affected by this problem.

The behavior of *cut* when provided with either mutually-exclusive options or options that do not work logically together has been deliberately left unspecified in favor of global wording in [Section 1.4](#) (on page 2288).

The OPTIONS section was changed in response to IEEE PASC Interpretation 1003.2 #149. The change represents historical practice on all known systems. The original standard was ambiguous on the nature of the output.

The *list* option-arguments are historically used to select the portions of the line to be written, but do not affect the order of the data. For example:

```
echo abcdefghi | cut -c6,2,4-7,1
```

yields "abdefg".

A proposal to enhance *cut* with the following option:

- o Preserve the selected field order. When this option is specified, each byte, character, or field (or ranges of such) shall be written in the order specified by the *list* option-argument, even if this requires multiple outputs of the same bytes, characters, or fields.

was rejected because this type of enhancement is outside the scope of the IEEE P1003.2b draft standard.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.5](#) (on page 2301), *fold*, *grep*, *paste*

[XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The OPTIONS section is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term “must” for application requirements.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-171 is applied, adding APPLICATION USAGE.

cxref

Utilities

82766 **NAME**82767 cxref — generate a C-language program cross-reference table (**DEVELOPMENT**)82768 **SYNOPSIS**

```
82769 xsi cxref [-cs] [-o file] [-w num] [-D name[=def]]... [-I dir]...
82770 [-U name]... file...
```

82771 **DESCRIPTION**

82772 The *cxref* utility shall analyze a collection of C-language *files* and attempt to build a cross-
 82773 reference table. Information from **#define** lines shall be included in the symbol table. A sorted
 82774 listing shall be written to standard output of all symbols (auto, static, and global) in each *file*
 82775 separately, or with the **-c** option, in combination. Each symbol shall contain an <asterisk> before
 82776 the declaring reference.

82777 **OPTIONS**

82778 The *cxref* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the order of the **-D**,
 82779 **-I**, and **-U** options (which are identical to their interpretation by *c99*) is significant. The
 82780 following options shall be supported:

- 82781 **-c** Write a combined cross-reference of all input files.
- 82782 **-s** Operate silently; do not print input filenames.
- 82783 **-o file** Direct output to named *file*.
- 82784 **-w num** Format output no wider than *num* (decimal) columns. This option defaults to 80 if
 82785 *num* is not specified or is less than 51.
- 82786 **-D** Equivalent to *c99*.
- 82787 **-I** Equivalent to *c99*.
- 82788 **-U** Equivalent to *c99*.

82789 **OPERANDS**

82790 The following operand shall be supported:

- 82791 *file* A pathname of a C-language source file.

82792 **STDIN**

82793 Not used.

82794 **INPUT FILES**

82795 The input files are C-language source files.

82796 **ENVIRONMENT VARIABLES**

82797 The following environment variables shall affect the execution of *cxref*:

- 82798 **LANG** Provide a default value for the internationalization variables that are unset or null.
 82799 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 82800 variables used to determine the values of locale categories.)
- 82801 **LC_ALL** If set to a non-empty string value, override the values of all the other
 82802 internationalization variables.
- 82803 **LC_COLLATE**
 82804 Determine the locale for the ordering of the output.
- 82805 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 82806 characters (for example, single-byte as opposed to multi-byte characters in
 82807 arguments and input files).

82808	LC_MESSAGES	
82809		Determine the locale that should be used to affect the format and contents of
82810		diagnostic messages written to standard error.
82811	NLSPATH	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
82812	ASYNCHRONOUS EVENTS	
82813		Default.
82814	STDOUT	
82815		The standard output shall be used for the cross-reference listing, unless the -o option is used to
82816		select a different output file.
82817		The format of standard output is unspecified, except that the following information shall be
82818		included:
82819		• If the -c option is not specified, each portion of the listing shall start with the name of the
82820		input file on a separate line.
82821		• The name line shall be followed by a sorted list of symbols, each with its associated
82822		location pathname, the name of the function in which it appears (if it is not a function
82823		name itself), and line number references.
82824		• Each line number may be preceded by an <asterisk> ('*') flag, meaning that this is the
82825		declaring reference. Other single-character flags, with implementation-defined meanings,
82826		may be included.
82827	STDERR	
82828		The standard error shall be used only for diagnostic messages.
82829	OUTPUT FILES	
82830		The output file named by the -o option shall be used instead of standard output.
82831	EXTENDED DESCRIPTION	
82832		None.
82833	EXIT STATUS	
82834		The following exit values shall be returned:
82835		0 Successful completion.
82836		>0 An error occurred.
82837	CONSEQUENCES OF ERRORS	
82838		Default.
82839	APPLICATION USAGE	
82840		None.
82841	EXAMPLES	
82842		None.
82843	RATIONALE	
82844		None.
82845	FUTURE DIRECTIONS	
82846		None.

82847 **SEE ALSO**82848 [c99](#)82849 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)82850 **CHANGE HISTORY**

82851 First released in Issue 2.

82852 **Issue 5**82853 In the SYNOPSIS, [-U *dir*] is changed to [-U *name*].82854 **Issue 6**

82855 The APPLICATION USAGE section is added.

82856 **Issue 7**

82857 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

date

82858 NAME

82859 date — write the date and time

82860 SYNOPSIS

82861 date [-u] [+format]

82862 XSI date [-u] mmddhhmm[[cc]yy]

82863 DESCRIPTION

82864 XSI The *date* utility shall write the date and time to standard output or attempt to set the system
82865 date and time. By default, the current date and time shall be written. If an operand beginning
82866 with '+' is specified, the output format of *date* shall be controlled by the conversion
82867 specifications and other text in the operand.

82868 OPTIONS

82869 The *date* utility shall conform to XBD Section 12.2 (on page 215).

82870 The following option shall be supported:

82871 -u Perform operations as if the *TZ* environment variable was set to the string "UTC0",
82872 or its equivalent historical value of "GMT0". Otherwise, *date* shall use the timezone
82873 indicated by the *TZ* environment variable or the system default if that variable is
82874 unset or null.

82875 OPERANDS

82876 The following operands shall be supported:

82877 +format When the format is specified, each conversion specifier shall be replaced in the
82878 standard output by its corresponding value. All other characters shall be copied to
82879 the output without change. The output shall always be terminated with a
82880 <newline>.

82881 Conversion Specifications

82882 %a Locale's abbreviated weekday name.

82883 %A Locale's full weekday name.

82884 %b Locale's abbreviated month name.

82885 %B Locale's full month name.

82886 %c Locale's appropriate date and time representation.

82887 %C Century (a year divided by 100 and truncated to an integer) as a decimal
82888 number [00,99].

82889 %d Day of the month as a decimal number [01,31].

82890 %D Date in the format *mm/dd/yy*.

82891 %e Day of the month as a decimal number [1,31] in a two-digit field with
82892 leading <space> character fill.

82893 %h A synonym for %b.

82894 %H Hour (24-hour clock) as a decimal number [00,23].

82895 %I Hour (12-hour clock) as a decimal number [01,12].

date

Utilities

82896	%j	Day of the year as a decimal number [001,366].
82897	%m	Month as a decimal number [01,12].
82898	%M	Minute as a decimal number [00,59].
82899	%n	A <newline>.
82900	%p	Locale's equivalent of either AM or PM.
82901	%r	12-hour clock time [01,12] using the AM/PM notation; in the POSIX locale, this shall be equivalent to %I:%M:%S %p.
82902		
82903	%S	Seconds as a decimal number [00,60].
82904	%t	A <tab>.
82905	%T	24-hour clock time [00,23] in the format <i>HH:MM:SS</i> .
82906	%u	Weekday as a decimal number [1,7] (1=Monday).
82907	%U	Week of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday shall be considered to be in week 0.
82908		
82909		
82910	%V	Week of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing January 1 has four or more days in the new year, then it shall be considered week 1; otherwise, it shall be the last week of the previous year, and the next week shall be week 1.
82911		
82912		
82913		
82914	%w	Weekday as a decimal number [0,6] (0=Sunday).
82915	%W	Week of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday shall be considered to be in week 0.
82916		
82917		
82918	%x	Locale's appropriate date representation.
82919	%X	Locale's appropriate time representation.
82920	%y	Year within century [00,99].
82921	%Y	Year with century as a decimal number.
82922	%Z	Timezone name, or no characters if no timezone is determinable.
82923	%%	A <percent-sign> character.
82924		See XBD Section 7.3.5 (on page 158) for the conversion specifier values in the POSIX locale.
82925		
82926		Modified Conversion Specifications
82927		Some conversion specifiers can be modified by the E and O modifier characters to indicate a different format or specification as specified in the <i>LC_TIME</i> locale description (see XBD Section 7.3.5 , on page 158). If the corresponding keyword (see era , era_year , era_d_fmt , and alt_digits in XBD Section 7.3.5 , on page 158) is not specified or not supported for the current locale, the unmodified conversion specifier value shall be used.
82928		
82929		
82930		
82931		
82932		
82933	%Ec	Locale's alternative appropriate date and time representation.

Utilities

date

82934	%EC	The name of the base year (period) in the locale's alternative representation.
82935		
82936	%Ex	Locale's alternative date representation.
82937	%EX	Locale's alternative time representation.
82938	%Ey	Offset from %EC (year only) in the locale's alternative representation.
82939	%EY	Full alternative year representation.
82940	%Od	Day of month using the locale's alternative numeric symbols.
82941	%Oe	Day of month using the locale's alternative numeric symbols.
82942	%OH	Hour (24-hour clock) using the locale's alternative numeric symbols.
82943	%OI	Hour (12-hour clock) using the locale's alternative numeric symbols.
82944	%Om	Month using the locale's alternative numeric symbols.
82945	%OM	Minutes using the locale's alternative numeric symbols.
82946	%OS	Seconds using the locale's alternative numeric symbols.
82947	%Ou	Weekday as a number in the locale's alternative representation (Monday = 1).
82948		
82949	%OU	Week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.
82950		
82951	%OV	Week number of the year (Monday as the first day of the week, rules corresponding to %V), using the locale's alternative numeric symbols.
82952		
82953	%Ow	Weekday as a number in the locale's alternative representation (Sunday = 0).
82954		
82955	%OW	Week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
82956		
82957	%Oy	Year (offset from %C) in alternative representation.
82958	XSI	mmddhhmm[[cc]yy]
82959		Attempt to set the system date and time from the value given in the operand. This
82960		is only possible if the user has appropriate privileges and the system permits the
82961		setting of the system date and time. The first <i>mm</i> is the month (number); <i>dd</i> is the
82962		day (number); <i>hh</i> is the hour (number, 24-hour system); the second <i>mm</i> is the
82963		minute (number); <i>cc</i> is the century and is the first two digits of the year (this is
82964		optional); <i>yy</i> is the last two digits of the year and is optional. If century is not
82965		specified, then values in the range [69,99] shall refer to years 1969 to 1999 inclusive,
82966		and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. The
82967		current year is the default if <i>yy</i> is omitted.
82968	Note:	It is expected that in a future version of this standard the default century inferred
82969		from a 2-digit year will change. (This would apply to all commands accepting a
82970		2-digit year as input.)
82971	STDIN	
82972		Not used.

date

Utilities

82973 **INPUT FILES**

82974 None.

82975 **ENVIRONMENT VARIABLES**82976 The following environment variables shall affect the execution of *date*:

82977 *LANG* Provide a default value for the internationalization variables that are unset or null.
 82978 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 82979 variables used to determine the values of locale categories.)

82980 *LC_ALL* If set to a non-empty string value, override the values of all the other
 82981 internationalization variables.

82982 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 82983 characters (for example, single-byte as opposed to multi-byte characters in
 82984 arguments).

82985 *LC_MESSAGES*

82986 Determine the locale that should be used to affect the format and contents of
 82987 diagnostic messages written to standard error.

82988 *LC_TIME* Determine the format and contents of date and time strings written by *date*.

82989 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

82990 *TZ* Determine the timezone in which the time and date are written, unless the *-u*
 82991 option is specified. If the *TZ* variable is unset or null and *-u* is not specified, an
 82992 unspecified system default timezone is used.

82993 **ASYNCHRONOUS EVENTS**

82994 Default.

82995 **STDOUT**

82996 When no formatting operand is specified, the output in the POSIX locale shall be equivalent to
 82997 specifying:

82998 *date* "+%a %b %e %H:%M-%S %Z %Y"82999 **STDERR**

83000 The standard error shall be used only for diagnostic messages.

83001 **OUTPUT FILES**

83002 None.

83003 **EXTENDED DESCRIPTION**

83004 None.

83005 **EXIT STATUS**

83006 The following exit values shall be returned:

83007 0 The date was written successfully.

83008 >0 An error occurred.

83009 **CONSEQUENCES OF ERRORS**

83010 Default.

APPLICATION USAGE

Conversion specifiers are of unspecified format when not in the POSIX locale. Some of them can contain <newline> characters in some locales, so it may be difficult to use the format shown in standard output for parsing the output of *date* in those locales.

The range of values for %S extends from 0 to 60 seconds to accommodate the occasional leap second.

Although certain of the conversion specifiers in the POSIX locale (such as the name of the month) are shown with initial capital letters, this need not be the case in other locales. Programs using these fields may need to adjust the capitalization if the output is going to be used at the beginning of a sentence.

The date string formatting capabilities are intended for use in Gregorian-style calendars, possibly with a different starting year (or years). The %x and %c conversion specifications, however, are intended for local representation; these may be based on a different, non-Gregorian calendar.

The %C conversion specification was introduced to allow a fallback for the %EC (alternative year format base year); it can be viewed as the base of the current subdivision in the Gregorian calendar. The century number is calculated as the year divided by 100 and truncated to an integer; it should not be confused with the use of ordinal numbers for centuries (for example, "twenty-first century"). Both the %EY and %y can then be viewed as the offset from %EC and %C, respectively.

The E and O modifiers modify the traditional conversion specifiers, so that they can always be used, even if the implementation (or the current locale) does not support the modifier.

The E modifier supports alternative date formats, such as the Japanese Emperor's Era, as long as these are based on the Gregorian calendar system. Extending the E modifiers to other date elements may provide an implementation-defined extension capable of supporting other calendar systems, especially in combination with the O modifier.

The O modifier supports time and date formats using the locale's alternative numerical symbols, such as Kanji or Hindi digits or ordinal number representation.

Non-European locales, whether they use Latin digits in computational items or not, often have local forms of the digits for use in date formats. This is not totally unknown even in Europe; a variant of dates uses Roman numerals for the months: the third day of September 1991 would be written as 3.IX.1991. In Japan, Kanji digits are regularly used for dates; in Arabic-speaking countries, Hindi digits are used. The %d, %e, %H, %I, %m, %S, %U, %w, %W, and %y conversion specifications always return the date and time field in Latin digits (that is, 0 to 9). The %O modifier was introduced to support the use for display purposes of non-Latin digits. In the LC_TIME category in *localedef*, the optional **alt_digits** keyword is intended for this purpose. As an example, assume the following (partial) *localedef* source:

```
alt_digits  " "; "I"; "II"; "III"; "IV"; "V"; "VI"; "VII"; "VIII" \
           "IX"; "X"; "XI"; "XII"
d_fmt      "%e.%Om.%Y"
```

With the above date, the command:

```
date "+%x"
```

would yield 3.IX.1991. With the same **d_fmt**, but without the **alt_digits**, the command would yield 3.9.1991.

EXAMPLES

1. The following are input/output examples of *date* used at arbitrary times in the POSIX locale:

```
$ date
Tue Jun 26 09:58:10 PDT 1990

$ date "+DATE: %m/%d/%y%nTIME: %H:%M:%S"
DATE: 11/02/91
TIME: 13:36:16
```

```
$ date "+TIME: %r"
TIME: 01:36:32 PM
```

2. Examples for Denmark, where the default date and time format is `%a %d %b %Y %T %Z`:

```
$ LANG=da_DK.iso_8859-1 date
ons 02 okt 1991 15:03:32 CET

$ LANG=da_DK.iso_8859-1 \
date "+DATO: %A den %e. %B %Y%nKLOKKEN: %H:%M:%S"
DATO: onsdag den 2. oktober 1991
KLOKKEN: 15:03:56
```

3. Examples for Germany, where the default date and time format is `%a %d.%h.%Y, %T %Z`:

```
$ LANG=De_DE.88591 date
Mi 02.Okt.1991, 15:01:21 MEZ

$ LANG=De_DE.88591 date "+DATUM: %A, %d. %B %Y%nZEIT: %H:%M:%S"
DATUM: Mittwoch, 02. Oktober 1991
ZEIT: 15:02:02
```

4. Examples for France, where the default date and time format is `%a %d %h %Y %Z %T`:

```
$ LANG=Fr_FR.88591 date
Mer 02 oct 1991 MET 15:03:32

$ LANG=Fr_FR.88591 date "+JOUR: %A %d %B %Y%nHEURE: %H:%M:%S"
JOUR: Mercredi 02 octobre 1991
HEURE: 15:03:56
```

RATIONALE

Some of the new options for formatting are from the ISO C standard. The `-u` option was introduced to allow portable access to Coordinated Universal Time (UTC). The string "GMT0" is allowed as an equivalent TZ value to be compatible with all of the systems using the BSD implementation, where this option originated.

The `%e` format conversion specification (adopted from System V) was added because the ISO C standard conversion specifications did not provide any way to produce the historical default *date* output during the first nine days of any month.

There are two varieties of day and week numbering supported (in addition to any others created with the locale-dependent `%E` and `%O` modifier characters):

- The historical variety in which Sunday is the first day of the week and the weekdays preceding the first Sunday of the year are considered week 0. These are represented by `%w` and `%U`. A variant of this is `%W`, using Monday as the first day of the week, but still referring to week 0. This view of the calendar was retained because so many historical

- 83098 applications depend on it and the ISO C standard *strftime()* function, on which many *date*
83099 implementations are based, was defined in this way.
- 83100 • The international standard, based on the ISO 8601:2004 standard where Monday is the first
83101 weekday and the algorithm for the first week number is more complex: If the week
83102 (Monday to Sunday) containing January 1 has four or more days in the new year, then it is
83103 week 1; otherwise, it is week 53 of the previous year, and the next week is week 1. These
83104 are represented by the new conversion specifications %u and %V, added as a result of
83105 international comments.
- 83106 **FUTURE DIRECTIONS**
- 83107 None.
- 83108 **SEE ALSO**
- 83109 XBD [Section 7.3.5](#) (on page 158), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)
- 83110 XSH [fprintf\(\)](#), [strftime\(\)](#)
- 83111 **CHANGE HISTORY**
- 83112 First released in Issue 2.
- 83113 **Issue 5**
- 83114 Changes are made for Year 2000 alignment.
- 83115 **Issue 6**
- 83116 The following new requirements on POSIX implementations derive from alignment with the
83117 Single UNIX Specification:
- 83118 • The %EX modified conversion specification is added.
- 83119 The Open Group Corrigendum U048/2 is applied, correcting the examples.
- 83120 The DESCRIPTION is updated to refer to conversion specifications, instead of field descriptors
83121 for consistency with the *LC_TIME* category.
- 83122 A clarification is made such that the current year is the default if the *yy* argument is omitted
83123 when setting the system date and time.
- 83124 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/19 is applied, correcting the CHANGE
83125 HISTORY section.

dd

Utilities

83126 **NAME**

83127 dd — convert and copy a file

83128 **SYNOPSIS**83129 dd [*operand*...]83130 **DESCRIPTION**

83131 The *dd* utility shall copy the specified input file to the specified output file with possible
 83132 conversions using specific input and output block sizes. It shall read the input one block at a
 83133 time, using the specified input block size; it shall then process the block of data actually
 83134 returned, which could be smaller than the requested block size. It shall apply any conversions
 83135 that have been specified and write the resulting data to the output in blocks of the specified
 83136 output block size. If the **bs=expr** operand is specified and no conversions other than **sync**,
 83137 **noerror**, or **notrunc** are requested, the data returned from each input block shall be written as a
 83138 separate output block; if the read returns less than a full block and the **sync** conversion is not
 83139 specified, the resulting output block shall be the same size as the input block. If the **bs=expr**
 83140 operand is not specified, or a conversion other than **sync**, **noerror**, or **notrunc** is requested, the
 83141 input shall be processed and collected into full-sized output blocks until the end of the input is
 83142 reached.

83143 The processing order shall be as follows:

- 83144 1. An input block is read.
- 83145 2. If the input block is shorter than the specified input block size and the **sync** conversion is
 83146 specified, null bytes shall be appended to the input data up to the specified size. (If either
 83147 **block** or **unblock** is also specified, <space> characters shall be appended instead of null
 83148 bytes.) The remaining conversions and output shall include the pad characters as if they
 83149 had been read from the input.
- 83150 3. If the **bs=expr** operand is specified and no conversion other than **sync** or **noerror** is
 83151 requested, the resulting data shall be written to the output as a single block, and the
 83152 remaining steps are omitted.
- 83153 4. If the **swab** conversion is specified, each pair of input data bytes shall be swapped. If
 83154 there is an odd number of bytes in the input block, the last byte in the input record shall
 83155 not be swapped.
- 83156 5. Any remaining conversions (**block**, **unblock**, **lcase**, and **ucase**) shall be performed. These
 83157 conversions shall operate on the input data independently of the input blocking; an input
 83158 or output fixed-length record may span block boundaries.
- 83159 6. The data resulting from input or conversion or both shall be aggregated into output
 83160 blocks of the specified size. After the end of input is reached, any remaining output shall
 83161 be written as a block without padding if **conv=sync** is not specified; thus, the final output
 83162 block may be shorter than the output block size.

83163 **OPTIONS**

83164 None.

83165 **OPERANDS**

83166 All of the operands shall be processed before any input is read. The following operands shall be
 83167 supported:

- 83168 **if=file** Specify the input pathname; the default is standard input.
- 83169 **of=file** Specify the output pathname; the default is standard output. If the **seek=expr**
 83170 conversion is not also specified, the output file shall be truncated before the copy
 83171 begins if an explicit **of=file** operand is specified, unless **conv=notrunc** is specified.

83172		If seek=expr is specified, but conv=notrunc is not, the effect of the copy shall be to preserve the blocks in the output file over which dd seeks, but no other portion of the output file shall be preserved. (If the size of the seek plus the size of the input file is less than the previous size of the output file, the output file shall be shortened by the copy. If the input file is empty and either the size of the seek is greater than the previous size of the output file or the output file did not previously exist, the size of the output file shall be set to the file offset after the seek.)
83173		
83174		
83175		
83176		
83177		
83178		
83179		
83180	ibs=expr	Specify the input block size, in bytes, by <i>expr</i> (default is 512).
83181	obs=expr	Specify the output block size, in bytes, by <i>expr</i> (default is 512).
83182	bs=expr	Set both input and output block sizes to <i>expr</i> bytes, superseding ibs= and obs= . If no conversion other than sync , noerror , and notrunc is specified, each input block shall be copied to the output as a single block without aggregating short blocks.
83183		
83184		
83185	cbs=expr	Specify the conversion block size for block and unblock in bytes by <i>expr</i> (default is zero). If cbs= is omitted or given a value of zero, using block or unblock produces unspecified results.
83186		
83187		
83188	XSI	The application shall ensure that this operand is also specified if the conv= operand is specified with a value of ascii , ebcdic , or ibm . For a conv= operand with an ascii value, the input is handled as described for the unblock value, except that characters are converted to ASCII before any trailing <space> characters are deleted. For conv= operands with ebcdic or ibm values, the input is handled as described for the block value except that the characters are converted to EBCDIC or IBM EBCDIC, respectively, after any trailing <space> characters are added.
83189		
83190		
83191		
83192		
83193		
83194		
83195	skip=n	Skip <i>n</i> input blocks (using the specified input block size) before starting to copy. On seekable files, the implementation shall read the blocks or seek past them; on non-seekable files, the blocks shall be read and the data shall be discarded.
83196		
83197		
83198	seek=n	Skip <i>n</i> blocks (using the specified output block size) from the beginning of the output file before copying. On non-seekable files, existing blocks shall be read and space from the current end-of-file to the specified offset, if any, filled with null bytes; on seekable files, the implementation shall seek to the specified offset or read the blocks as described for non-seekable files.
83199		
83200		
83201		
83202		
83203	count=n	Copy only <i>n</i> input blocks.
83204	conv=value[,value ...]	
83205		Where <i>values</i> are <comma>-separated symbols from the following list:
83206	XSI	ascii Convert EBCDIC to ASCII; see Table 4-7 (on page 2585).
83207	XSI	ebcdic Convert ASCII to EBCDIC; see Table 4-7 (on page 2585).
83208	XSI	ibm Convert ASCII to a different EBCDIC set; see Table 4-8 (on page 2586).
83209		
83210	XSI	The ascii , ebcdic , and ibm values are mutually-exclusive.
83211	block	Treat the input as a sequence of <newline>-terminated or end-of-file-terminated variable-length records independent of the input block boundaries. Each record shall be converted to a record with a fixed length specified by the conversion block size. Any <newline> shall be removed from the input line; <space> characters shall be appended to lines that are shorter than their conversion block size to fill the block.
83212		
83213		
83214		
83215		
83216		

dd*Utilities*

83217		Lines that are longer than the conversion block size shall be truncated
83218		to the largest number of characters that fit into that size; the number of
83219		truncated lines shall be reported (see the STDERR section).
83220		The block and unblock values are mutually-exclusive.
83221	unblock	Convert fixed-length records to variable length. Read a number of bytes
83222		equal to the conversion block size (or the number of bytes remaining in
83223		the input, if less than the conversion block size), delete all trailing
83224		<space> characters, and append a <newline>.
83225	lcase	Map uppercase characters specified by the <i>LC_CTYPE</i> keyword
83226		tolower to the corresponding lowercase character. Characters for which
83227		no mapping is specified shall not be modified by this conversion.
83228		The lcase and ucase symbols are mutually-exclusive.
83229	ucase	Map lowercase characters specified by the <i>LC_CTYPE</i> keyword
83230		toupper to the corresponding uppercase character. Characters for
83231		which no mapping is specified shall not be modified by this conversion.
83232	swab	Swap every pair of input bytes.
83233	noerror	Do not stop processing on an input error. When an input error occurs, a
83234		diagnostic message shall be written on standard error, followed by the
83235		current input and output block counts in the same format as used at
83236		completion (see the STDERR section). If the sync conversion is
83237		specified, the missing input shall be replaced with null bytes and
83238		processed normally; otherwise, the input block shall be omitted from
83239		the output.
83240	notrunc	Do not truncate the output file. Preserve blocks in the output file not
83241		explicitly written by this invocation of the <i>dd</i> utility. (See also the
83242		preceding of=file operand.)
83243	sync	Pad every input block to the size of the ibs= buffer, appending null
83244		bytes. (If either block or unblock is also specified, append <space>
83245		characters, rather than null bytes.)
83246		The behavior is unspecified if operands other than conv= are specified more than once.
83247		For the bs= , cbs= , ibs= , and obs= operands, the application shall supply an expression
83248		specifying a size in bytes. The expression, <i>expr</i> , can be:
83249		1. A positive decimal number
83250		2. A positive decimal number followed by <i>k</i> , specifying multiplication by 1 024
83251		3. A positive decimal number followed by <i>b</i> , specifying multiplication by 512
83252		4. Two or more positive decimal numbers (with or without <i>k</i> or <i>b</i>) separated by <i>x</i> , specifying
83253		the product of the indicated values
83254		All of the operands are processed before any input is read.
83255	XSI	The following two tables display the octal number character values used for the ascii and ebcdic
83256		conversions (first table) and for the ibm conversion (second table). In both tables, the ASCII
83257		values are the row and column headers and the EBCDIC values are found at their intersections.
83258		For example, ASCII 0012 (LF) is the second row, third column, yielding 0045 in EBCDIC. The
83259		inverted tables (for EBCDIC to ASCII conversion) are not shown, but are in one-to-one

83260
83261

correspondence with these tables. The differences between the two tables are highlighted by small boxes drawn around five entries.

83262

Table 4-7 ASCII to EBCDIC Conversion

0	1	2	3	4	5	6	7
0000 NUL	0001 SOH	0002 STX	0003 ETX	0067 EOT	0055 ENQ	0056 ACK	0057 BEL
0026 BS	0005 HT	0045 LF	0013 VT	0014 FF	0015 CR	0016 SO	0017 SI
0020 DLE	0021 DC1	0022 DC2	0023 DC3	0074 DC4	0075 NAK	0062 SYN	0046 ETB
0030 CAN	0031 EM	0077 SUB	0047 ESC	0034 IFS	0035 IGS	0036 IRS	0037 ITB
0100 Sp	0132 !	0177 "	0173 #	0133 \$	0154 %	0120 &	0175 .
0115 (	0135)	0134 *	0116 +	0153 ,	0140 -	0113 .	0141 /
0360 0	0361 1	0362 2	0363 3	0364 4	0365 5	0366 6	0367 7
0370 8	0371 9	0172 :	0136 ;	0114 <	0176 =	0156 >	0157 ?
0174 @	0301 A	0302 B	0303 C	0304 D	0305 E	0306 F	0307 G
0310 H	0311 I	0321 J	0322 K	0323 L	0324 M	0325 N	0326 O
0327 P	0330 Q	0331 R	0342 S	0343 T	0344 U	0345 V	0346 W
0347 X	0350 Y	0351 Z	0255 [	0340 \	0275]	0232 ^	0155 _
0171 ,	0201 a	0202 b	0203 c	0204 d	0205 e	0206 f	0207 g
0210 h	0211 i	0221 j	0222 k	0223 l	0224 m	0225 n	0226 o
0227 p	0230 q	0231 r	0242 s	0243 t	0244 u	0245 v	0246 w
0247 x	0250 y	0251 z	0300 {	0117	0320 }	0137 ~	0007 DEL
0040 DS	0041 SOS	0042 FS	0043 WUS	0044 BYP	0025 NL	0006 RNL	0027 POC
0050 SA	0051 SFE	0052 SM	0053 CSP	0054 MFA	0011 SPS	0012 RPT	0033 CU1
0060	0061	0032 UBS	0063 IR	0064 PP	0065 TRN	0066 NBS	0010 GE
0070 SBS	0071 IT	0072 RFF	0073 CU3	0004 SEL	0024 RES	0076	00341
0101	0102	0103	0104	0105	0106	0107	0110
0111	0121	0122	0123	0124	0125	0126	0127
0130	0131	0142	0143	0144	0145	0146	0147
0150	0151	0160	0161	0162	0163	0164	0165
0166	0167	0170	0200	0212	0213	0214	0215
0216	0217	0220	0152 ¡	0233	0234	0235	0236
0237	0240	0252	0253	0254	0112 ¢	0256	0257
0260	0261	0262	0263	0264	0265	0266	0267
0270	0271	0272	0273	0274	0241	0276	0277
0312	0313	0314 ¢	0315	0316 ¢	0317	0332	0333
0334	0335	0336	0337	0352	0353	0354	0355
0356	0357	0372	0373	0374	0375	0376	0377 EO

dd

Utilities

83263

Table 4-8 ASCII to IBM EBCDIC Conversion

	0	1	2	3	4	5	6	7								
0000	0000	NUL	0001	SOH	0002	STX	0003	ETX	0004	EOT	0005	ENQ	0006	ACK	0007	BEL
0010	0026	BS	0005	HT	0045	LF	0013	VT	0014	FF	0015	CR	0016	SO	0017	SI
0020	0020	DLE	0021	DC1	0022	DC2	0023	DC3	0074	DC4	0075	NAK	0062	SYN	0046	ETB
0030	0030	CAN	0031	EM	0077	SUB	0047	ESC	0034	IFS	0035	IGS	0036	IRS	0037	ITB
0040	0100	Sp	0132	!	0177	"	0173	#	0133	\$	0154	%	0120	&	0175	,
0050	0115	(	0135	)	0134	*	0116	+	0153	,	0140	-	0113	.	0141	/
0060	0360	0	0361	1	0362	2	0363	3	0364	4	0365	5	0366	6	0367	7
0070	0370	8	0371	9	0172	:	0136	;	0114	<	0176	=	0156	>	0157	?
0100	0174	@	0301	A	0302	B	0303	C	0304	D	0305	E	0306	F	0307	G
0110	0310	H	0311	I	0321	J	0322	K	0323	L	0324	M	0325	N	0326	O
0120	0327	P	0330	Q	0331	R	0342	S	0343	T	0344	U	0345	V	0346	W
0130	0347	X	0350	Y	0351	Z	0255	[	0340	\	0275	]	0137	^	0155	_
0140	0171	,	0201	a	0202	b	0203	c	0204	d	0205	e	0206	f	0207	g
0150	0210	h	0211	i	0221	j	0222	k	0223	l	0224	m	0225	n	0226	o
0160	0227	p	0230	q	0231	r	0242	s	0243	t	0244	u	0245	v	0246	w
0170	0247	x	0250	y	0251	z	0300	{	0117		0320	}	0241	~	0007	DEL
0200	0040	DS	0041	SOS	0042	FS	0043	WUS	0044	BYP	0025	NL	0006	RNL	0027	POC
0210	0050	SA	0051	SFE	0052	SM	0053	CSP	0054	MFA	0011	SPS	0012	RPT	0033	CU1
0220	0060		0061		0032	UBS	0063	IR	0064	PP	0065	TRN	0066	NBS	0010	GE
0230	0070	SBS	0071	IT	0072	RFF	0073	CU3	0004	SEL	0024	RES	0076		0341	
0240	0101		0102		0103		0104		0105		0106		0107		0110	
0250	0111		0121		0122		0123		0124		0125		0126		0127	
0260	0130		0131		0142		0143		0144		0145		0146		0147	
0270	0150		0151		0160		0161		0162		0163		0164		0165	
0300	0166		0167		0170		0200		0212		0213		0214		0215	
0310	0216		0217		0220		0232		0233		0234		0235		0236	
0320	0237		0240		0252		0253		0254		0255	[	0256		0257	
0330	0260		0261		0262		0263		0264		0265		0266		0267	
0340	0270		0271		0272		0273		0274		0275	]	0276		0277	
0350	0312		0313		0314	␣	0315		0316	␣	0317		0332		0333	
0360	0334		0335		0336		0337		0352		0353		0354	␣	0355	
0370	0356		0357		0372		0373		0374		0375		0376		0377	EO

83264 **STDIN**

83265 If no **if=** operand is specified, the standard input shall be used. See the INPUT FILES section.

83266 **INPUT FILES**

83267 The input file can be any file type.

83268 **ENVIRONMENT VARIABLES**

83269 The following environment variables shall affect the execution of *dd*:

83270 **LANG** Provide a default value for the internationalization variables that are unset or null.
 83271 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 83272 variables used to determine the values of locale categories.)

83273 **LC_ALL** If set to a non-empty string value, override the values of all the other
 83274 internationalization variables.

83275 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 83276 characters (for example, single-byte as opposed to multi-byte characters in
 83277 arguments and input files), the classification of characters as uppercase or
 83278 lowercase, and the mapping of characters from one case to the other.

83279 **LC_MESSAGES**

83280 Determine the locale that should be used to affect the format and contents of
 83281 diagnostic messages written to standard error and informative messages written to
 83282 standard output.

83283 **XSI NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

83284 **ASYNCHRONOUS EVENTS**

83285 For SIGINT, the *dd* utility shall interrupt its current processing, write status information to
 83286 standard error, and exit as though terminated by SIGINT. It shall take the standard action for all
 83287 other signals; see the ASYNCHRONOUS EVENTS section in Section 1.4 (on page 2288).

83288 **STDOUT**

83289 If no **of=** operand is specified, the standard output shall be used. The nature of the output
 83290 depends on the operands selected.

83291 **STDERR**

83292 On completion, *dd* shall write the number of input and output blocks to standard error. In the
 83293 POSIX locale the following formats shall be used:

83294 "%u+%u records in\n", <number of whole input blocks>,
 83295 <number of partial input blocks>

83296 "%u+%u records out\n", <number of whole output blocks>,
 83297 <number of partial output blocks>

83298 A partial input block is one for which *read()* returned less than the input block size. A partial
 83299 output block is one that was written with fewer bytes than specified by the output block size.

83300 In addition, when there is at least one truncated block, the number of truncated blocks shall be
 83301 written to standard error. In the POSIX locale, the format shall be:

83302 "%u truncated %s\n", <number of truncated blocks>, "record" (if
 83303 <number of truncated blocks> is one) "records" (otherwise)

83304 Diagnostic messages may also be written to standard error.

83305 **OUTPUT FILES**

83306 If the **of=** operand is used, the output shall be the same as described in the STDOUT section.

83307 **EXTENDED DESCRIPTION**

83308 None.

83309 **EXIT STATUS**

83310 The following exit values shall be returned:

83311 0 The input file was copied successfully.

83312 >0 An error occurred.

83313 **CONSEQUENCES OF ERRORS**

83314 If an input error is detected and the **noerror** conversion has not been specified, any partial
83315 output block shall be written to the output file, a diagnostic message shall be written, and the
83316 copy operation shall be discontinued. If some other error is detected, a diagnostic message shall
83317 be written and the copy operation shall be discontinued.

83318 **APPLICATION USAGE**

83319 The input and output block size can be specified to take advantage of raw physical I/O.

83320 There are many different versions of the EBCDIC codesets. The ASCII and EBCDIC conversions
83321 specified for the *dd* utility perform conversions for the version specified by the tables.

83322 **EXAMPLES**

83323 The following command:

83324 `dd if=/dev/rmt0h of=/dev/rmt1h`

83325 copies from tape drive 0 to tape drive 1, using a common historical device naming convention.

83326 The following command:

83327 `dd ibs=10 skip=1`

83328 strips the first 10 bytes from standard input.

83329 This example reads an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the
83330 ASCII file *x*:

83331 `dd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase`

83332 **RATIONALE**

83333 The OPTIONS section is listed as “None” because there are no options recognized by historical
83334 *dd* utilities. Certainly, many of the operands could have been designed to use the Utility Syntax
83335 Guidelines, which would have resulted in the classic hyphenated option letters. In this version
83336 of this volume of POSIX.1-2008, *dd* retains its curious JCL-like syntax due to the large number of
83337 applications that depend on the historical implementation.

83338 A suggested implementation technique for **conv=noerror, sync** is to zero (or <space>-fill, if
83339 **blocking** or **unblocking**) the input buffer before each read and to write the contents of the input
83340 buffer to the output even after an error. In this manner, any data transferred to the input buffer
83341 before the error was detected is preserved. Another point is that a failed read on a regular file or
83342 a disk generally does not increment the file offset, and *dd* must then seek past the block on which
83343 the error occurred; otherwise, the input error occurs repetitively. When the input is a magnetic
83344 tape, however, the tape normally has passed the block containing the error when the error is
83345 reported, and thus no seek is necessary.

83346 The default **ibs=** and **obs=** sizes are specified as 512 bytes because there are historical (largely
83347 portable) scripts that assume these values. If they were left unspecified, unusual results could

occur if an implementation chose an odd block size.

Historical implementations of *dd* used *creat()* when processing *of=file*. This makes the *seek=* operand unusable except on special files. The *conv=notrunc* feature was added because more recent BSD-based implementations use *open()* (without *O_TRUNC*) instead of *creat()*, but they fail to delete output file contents after the data copied.

The *w* multiplier (historically meaning *word*), is used in System V to mean 2 and in 4.2 BSD to mean 4. Since *word* is inherently non-portable, its use is not supported by this volume of POSIX.1-2008.

Standard EBCDIC does not have the characters ' [' and '] '. The values used in the table are taken from a common print train that does contain them. Other than those characters, the print train values are not filled in, but appear to provide some of the motivation for the historical choice of translations reflected here.

The Standard EBCDIC table provides a 1:1 translation for all 256 bytes.

The IBM EBCDIC table does not provide such a translation. The marked cells in the tables differ in such a way that:

1. EBCDIC 0112 (' ¢ ') and 0152 (broken pipe) do not appear in the table.
2. EBCDIC 0137 (' ¬ ') translates to/from ASCII 0236 (' ^ '). In the standard table, EBCDIC 0232 (no graphic) is used.
3. EBCDIC 0241 (' ~ ') translates to/from ASCII 0176 (' ` '). In the standard table, EBCDIC 0137 (' ¬ ') is used.
4. 0255 (' [') and 0275 ('] ') appear twice, once in the same place as for the standard table and once in place of 0112 (' ¢ ') and 0241 (' ~ ').

In net result:

EBCDIC 0275 ('] ') displaced EBCDIC 0241 (' ~ ') in cell 0345.

That displaced EBCDIC 0137 (' ¬ ') in cell 0176.

That displaced EBCDIC 0232 (no graphic) in cell 0136.

That replaced EBCDIC 0152 (broken pipe) in cell 0313.

EBCDIC 0255 (' [') replaced EBCDIC 0112 (' ¢ ').

This translation, however, reflects historical practice that (ASCII) ' ^ ' and ' ¬ ' were often mapped to each other, as were ' [' and ' ¢ ' ; and '] ' and (EBCDIC) ' ~ ' .

The *cbs* operand is required if any of the *ascii*, *ebcdic*, or *ibm* operands are specified. For the *ascii* operand, the input is handled as described for the *unblock* operand except that characters are converted to ASCII before the trailing <space> characters are deleted. For the *ebcdic* and *ibm* operands, the input is handled as described for the *block* operand except that the characters are converted to EBCDIC or IBM EBCDIC after the trailing <space> characters are added.

The *block* and *unblock* keywords are from historical BSD practice.

The consistent use of the word *record* in standard error messages matches most historical practice. An earlier version of System V used *block*, but this has been updated in more recent releases.

Early proposals only allowed two numbers separated by *x* to be used in a product when specifying *bs=*, *cbs=*, *ibs=*, and *obs=* sizes. This was changed to reflect the historical practice of

- 83389 allowing multiple numbers in the product as provided by Version 7 and all releases of System V
83390 and BSD.
- 83391 A change to the **swab** conversion is required to match historical practice and is the result of IEEE
83392 PASC Interpretations 1003.2 #03 and #04, submitted for the ISO POSIX-2: 1993 standard.
- 83393 A change to the handling of SIGINT is required to match historical practice and is the result of
83394 IEEE PASC Interpretation 1003.2 #06 submitted for the ISO POSIX-2: 1993 standard.
- 83395 **FUTURE DIRECTIONS**
- 83396 None.
- 83397 **SEE ALSO**
- 83398 [Section 1.4](#) (on page 2288), *sed*, *tr*
- 83399 XBD [Chapter 8](#) (on page 173)
- 83400 **CHANGE HISTORY**
- 83401 First released in Issue 2.
- 83402 **Issue 5**
- 83403 The second paragraph of the **cbs=** description is reworded and marked EX.
- 83404 The FUTURE DIRECTIONS section is added.
- 83405 **Issue 6**
- 83406 Changes are made to **swab** conversion and SIGINT handling to align with the IEEE P1003.2b
83407 draft standard.
- 83408 The normative text is reworded to avoid use of the term “must” for application requirements.
- 83409 IEEE PASC Interpretation 1003.2 #209 is applied, clarifying the interaction between **dd of=file** and
83410 **conv=notrunc**.
- 83411 **Issue 7**
- 83412 Austin Group Interpretation 1003.1-2001 #102 is applied.
- 83413 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Utilities

delta

NAME

delta — make a delta (change) to an SCCS file (**DEVELOPMENT**)

SYNOPSIS

```
delta [-nps] [-g list] [-m mrlist] [-r SID] [-y[comment]] file...
```

DESCRIPTION

The *delta* utility shall be used to permanently introduce into the named SCCS files changes that were made to the files retrieved by *get* (called the *g-files*, or generated files).

OPTIONS

The *delta* utility shall conform to XBD Section 12.2 (on page 215), except that the *-y* option has an optional option-argument. This optional option-argument shall not be presented as a separate argument.

The following options shall be supported:

-r SID Uniquely identify which delta is to be made to the SCCS file. The use of this option shall be necessary only if two or more outstanding *get* commands for editing (*get -e*) on the same SCCS file were done by the same person (login name). The SID value specified with the *-r* option can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* utility; see *get* (on page 2764).

-s Suppress the report to standard output of the activity associated with each *file*. See the STDOUT section.

-n Specify retention of the edited *g-file* (normally removed at completion of delta processing).

-g list Specify a *list* (see *get* for the definition of *list*) of deltas that shall be ignored when the file is accessed at the change level (SID) created by this delta.

-m mrlist Specify a modification request (MR) number that the application shall supply as the reason for creating the new delta. This shall be used if the SCCS file has the *v* flag set; see *admin*.

If *-m* is not used and *'-'* is not specified as a file argument, and the standard input is a terminal, the prompt described in the STDOUT section shall be written to standard output before the standard input is read; if the standard input is not a terminal, no prompt shall be issued.

MRs in a list shall be separated by <blank> characters or escaped <newline> characters. An unescaped <newline> shall terminate the MR list. The escape character is <backslash>.

If the *v* flag has a value, it shall be taken to be the name of a program which validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, the *delta* utility shall terminate. (It is assumed that the MR numbers were not all valid.)

-y[comment] Describe the reason for making the delta. The *comment* shall be an arbitrary group of lines that would meet the definition of a text file. Implementations shall support *comments* from zero to 512 bytes and may support longer values. A null string (specified as either *-y*, *-y"*, or in response to a prompt for a comment) shall be considered a valid *comment*.

If *-y* is not specified and *'-'* is not specified as a file argument, and the standard

delta*Utilities*

83458 input is a terminal, the prompt described in the STDOUT section shall be written
 83459 to standard output before the standard input is read; if the standard input is not a
 83460 terminal, no prompt shall be issued. An unescaped <newline> shall terminate the
 83461 comment text. The escape character is <backslash>.

83462 The **-y** option shall be required if the *file* operand is specified as ' - '.

83463 **-p** Write (to standard output) the SCCS file differences before and after the delta is
 83464 applied in *diff* format; see *diff*.

OPERANDS

83465 The following operand shall be supported:

83467 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *delta*
 83468 utility shall behave as though each file in the directory were specified as a named
 83469 file, except that non-SCCS files (last component of the pathname does not begin
 83470 with **s.**) and unreadable files shall be silently ignored.

83471 If exactly one *file* operand appears, and it is ' - ', the standard input shall be read;
 83472 each line of the standard input shall be taken to be the name of an SCCS file to be
 83473 processed. Non-SCCS files and unreadable files shall be silently ignored.

STDIN

83474 The standard input shall be a text file used only in the following cases:

- 83476 • To read an *mrlist* or a *comment* (see the **-m** and **-y** options).
- 83477 • A *file* operand shall be specified as ' - '. In this case, the **-y** option must be used to specify
 83478 the comment, and if the SCCS file has the **v** flag set, the **-m** option must also be used to
 83479 specify the MR list.

INPUT FILES

83481 Input files shall be text files whose data is to be included in the SCCS files. If the first character of
 83482 any line of an input file is <SOH> in the POSIX locale, the results are unspecified. If this file
 83483 contains more than 99 999 lines, the number of lines recorded in the header for this file shall be
 83484 99 999 for this delta.

ENVIRONMENT VARIABLES

83485 The following environment variables shall affect the execution of *delta*:

83487 **LANG** Provide a default value for the internationalization variables that are unset or null.
 83488 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 83489 variables used to determine the values of locale categories.)

83490 **LC_ALL** If set to a non-empty string value, override the values of all the other
 83491 internationalization variables.

83492 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 83493 characters (for example, single-byte as opposed to multi-byte characters in
 83494 arguments and input files).

83495 **LC_MESSAGES**

83496 Determine the locale that should be used to affect the format and contents of
 83497 diagnostic messages written to standard error, and informative messages written
 83498 to standard output.

83499 **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

83500 **TZ** Determine the timezone in which the time and date are written in the SCCS file. If
 83501 the *TZ* variable is unset or NULL, an unspecified system default timezone is used.

83502 **ASYNCHRONOUS EVENTS**

83503 If SIGINT is caught, temporary files shall be cleaned up and *delta* shall exit with a non-zero exit
 83504 code. The standard action shall be taken for all other signals; see [Section 1.4](#) (on page 2288).

83505 **STDOUT**

83506 The standard output shall be used only for the following messages in the POSIX locale:

- 83507 • Prompts (see the *-m* and *-y* options) in the following formats:

83508 "MRs? "

83509 "comments? "

83510 The MR prompt, if written, shall always precede the comments prompt.

- 83511 • A report of each file's activities (unless the *-s* option is specified) in the following format:

83512 "%s\n%d inserted\n%d deleted\n%d unchanged\n", <New SID>,
 83513 <number of lines inserted>, <number of lines deleted>,
 83514 <number of lines unchanged>

83515 **STDERR**

83516 The standard error shall be used only for diagnostic messages.

83517 **OUTPUT FILES**

83518 Any SCCS files updated shall be files of an unspecified format.

83519 **EXTENDED DESCRIPTION**

83520 **System Date and Time**

83521 When a *delta* is added to an SCCS file, the system date and time shall be recorded for the new
 83522 delta. If a *get* is performed using an SCCS file with a date recorded apparently in the future, the
 83523 behavior is unspecified.

83524 **EXIT STATUS**

83525 The following exit values shall be returned:

83526 0 Successful completion.

83527 >0 An error occurred.

83528 **CONSEQUENCES OF ERRORS**

83529 Default.

83530 **APPLICATION USAGE**

83531 Problems can arise if the system date and time have been modified (for example, put forward
 83532 and then back again, or unsynchronized clocks across a network) and can also arise when
 83533 different values of the *TZ* environment variable are used.

83534 Problems of a similar nature can also arise for the operation of the *get* utility, which records the
 83535 date and time in the file body.

83536 **EXAMPLES**

83537 None.

83538 **RATIONALE**

83539 None.

83540 **FUTURE DIRECTIONS**

83541 None.

83542 **SEE ALSO**83543 [Section 1.4](#) (on page 2288), *admin*, *diff*, *get*, *prs*, *rmDEL*83544 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)83545 **CHANGE HISTORY**

83546 First released in Issue 2.

83547 **Issue 5**

83548 The output format description in the STDOUT section is corrected.

83549 **Issue 6**

83550 The APPLICATION USAGE section is added.

83551 The normative text is reworded to avoid use of the term “must” for application requirements.

83552 The Open Group Base Resolution bwg2001-007 is applied as follows:

- 83553 • The use of ‘-’ as a file argument is clarified.
- 83554 • The use of STDIN is added.
- 83555 • The ASYNCHRONOUS EVENTS section is updated to remove the implicit requirement that implementations re-signal themselves when catching a normally fatal signal.
- 83556 • The ASYNCHRONOUS EVENTS section is updated to remove the implicit requirement that implementations re-signal themselves when catching a normally fatal signal.
- 83557 • New text is added to the INPUT FILES section warning that the maximum lines recorded in the file is 99 999.
- 83558 • New text is added to the INPUT FILES section warning that the maximum lines recorded in the file is 99 999.

83559 New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections regarding how the system date and time may be taken into account, and the TZ environment variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base Resolution bwg2001-007.

83560

83561

83562

83563 **Issue 7**

83564 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Utilities

df

NAME

df — report free disk space

SYNOPSIS

df [-k] [-P|-t] [*file*...]

DESCRIPTION

The *df* utility shall write the amount of available space and file slots for file systems on which the invoking user has appropriate read access. File systems shall be specified by the *file* operands; when none are specified, information shall be written for all file systems. The format of the default output from *df* is unspecified, but all space figures are reported in 512-byte units, unless the *-k* option is specified. This output shall contain at least the file system names, amount of available space on each of these file systems, and the number of free file slots, or *inodes*, available; when *-t* is specified, the output shall contain the total allocated space as well.

OPTIONS

The *df* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

-k Use 1 024-byte units, instead of the default 512-byte units, when writing space figures.

-P Produce output in the format described in the STDOUT section.

-t Include total allocated-space figures in the output.

OPERANDS

The following operand shall be supported:

file A pathname of a file within the hierarchy of the desired file system. If a file other than a FIFO, a regular file, a directory, or a special file representing the device containing the file system (for example, */dev/dsk/0s1*) is specified, the results are unspecified. If the *file* operand names a file other than a special file containing a file system, *df* shall write the amount of free space in the file system containing the specified *file* operand. Otherwise, *df* shall write the amount of free space in that file system.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *df*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

83607	<i>LC_MESSAGES</i>	
83608		Determine the locale that should be used to affect the format and contents of
83609		diagnostic messages written to standard error and informative messages written to
83610		standard output.
83611	XSI	<i>NLSPATH</i> Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
83612	ASYNCHRONOUS EVENTS	
83613		Default.
83614	STDOUT	
83615		When both the -k and -P options are specified, the following header line shall be written (in the
83616		POSIX locale):
83617		"Filesystem 1024-blocks Used Available Capacity Mounted on\n"
83618		When the -P option is specified without the -k option, the following header line shall be written
83619		(in the POSIX locale):
83620		"Filesystem 512-blocks Used Available Capacity Mounted on\n"
83621		The implementation may adjust the spacing of the header line and the individual data lines so
83622		that the information is presented in orderly columns.
83623		The remaining output with -P shall consist of one line of information for each specified file
83624		system. These lines shall be formatted as follows:
83625		"%s %d %d %d %d%% %s\n", <file system name>, <total space>,
83626		<space used>, <space free>, <percentage used>,
83627		<file system root>
83628		In the following list, all quantities expressed in 512-byte units (1 024-byte when -k is specified)
83629		shall be rounded up to the next higher unit. The fields are:
83630		<file system name>
83631		The name of the file system, in an implementation-defined format.
83632		<total space> The total size of the file system in 512-byte units. The exact meaning of this figure
83633		is implementation-defined, but should include <space used>, <space free>, plus any
83634		space reserved by the system not normally available to a user.
83635		<space used> The total amount of space allocated to existing files in the file system, in 512-byte
83636		units.
83637		<space free> The total amount of space available within the file system for the creation of new
83638		files by unprivileged users, in 512-byte units. When this figure is less than or equal
83639		to zero, it shall not be possible to create any new files on the file system without
83640		first deleting others, unless the process has appropriate privileges. The figure
83641		written may be less than zero.
83642		<percentage used>
83643		The percentage of the normally available space that is currently allocated to all files
83644		on the file system. This shall be calculated using the fraction:
83645		<space used>/ (<space used>+ <space free>)
83646		expressed as a percentage. This percentage may be greater than 100 if <space free>
83647		is less than zero. The percentage value shall be expressed as a positive integer, with
83648		any fractional result causing it to be rounded to the next highest integer.

Utilities

df

83649 <file system root>
 83650 The directory below which the file system hierarchy appears.

83651 XSI The output format is unspecified when **-t** is used.

83652 **STDERR**

83653 The standard error shall be used only for diagnostic messages.

83654 **OUTPUT FILES**

83655 None.

83656 **EXTENDED DESCRIPTION**

83657 None.

83658 **EXIT STATUS**

83659 The following exit values shall be returned:

83660 0 Successful completion.

83661 >0 An error occurred.

83662 **CONSEQUENCES OF ERRORS**

83663 Default.

83664 **APPLICATION USAGE**

83665 On most systems, the “name of the file system, in an implementation-defined format” is the
 83666 special file on which the file system is mounted.

83667 On large file systems, the calculation specified for percentage used can create huge rounding
 83668 errors.

83669 **EXAMPLES**

83670 1. The following example writes portable information about the **/usr** file system:

83671 `df -P /usr`

83672 2. Assuming that **/usr/src** is part of the **/usr** file system, the following produces the same
 83673 output as the previous example:

83674 `df -P /usr/src`

83675 **RATIONALE**

83676 The behavior of **df** with the **-P** option is the default action of the 4.2 BSD **df** utility. The uppercase
 83677 **-P** was selected to avoid collision with a known industry extension using **-p**.

83678 Historical **df** implementations vary considerably in their default output. It was therefore
 83679 necessary to describe the default output in a loose manner to accommodate all known historical
 83680 implementations and to add a portable option (**-P**) to provide information in a portable format.

83681 The use of 512-byte units is historical practice and maintains compatibility with **ls** and other
 83682 utilities in this volume of POSIX.1-2008. This does not mandate that the file system itself be
 83683 based on 512-byte blocks. The **-k** option was added as a compromise measure. It was agreed by
 83684 the standard developers that 512 bytes was the best default unit because of its complete
 83685 historical consistency on System V (*versus* the mixed 512/1 024-byte usage on BSD systems), and
 83686 that a **-k** option to switch to 1 024-byte units was a good compromise. Users who prefer the
 83687 more logical 1 024-byte quantity can easily alias **df** to **df -k** without breaking many historical
 83688 scripts relying on the 512-byte units.

83689 It was suggested that **df** and the various related utilities be modified to access a **BLOCKSIZE**
 83690 environment variable to achieve consistency and user acceptance. Since this is not historical

df*Utilities*

83691 practice on any system, it is left as a possible area for system extensions and will be re-evaluated
83692 in a future version if it is widely implemented.

83693 FUTURE DIRECTIONS

83694 None.

83695 SEE ALSO

83696 *find*

83697 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

83698 CHANGE HISTORY

83699 First released in Issue 2.

83700 Issue 6

83701 This utility is marked as part of the User Portability Utilities option.

83702 Issue 7

83703 Austin Group Interpretation 1003.1-2001 #099 is applied.

83704 The *df* utility is removed from the User Portability Utilities option. User Portability Utilities is
83705 now an option for interactive utilities.

83706 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

diff**NAME**

diff — compare two files

SYNOPSIS

diff [-c|-e|-f|-u|-C *n*|-U *n*] [-br] *file1 file2*

DESCRIPTION

The *diff* utility shall compare the contents of *file1* and *file2* and write to standard output a list of changes necessary to convert *file1* into *file2*. This list should be minimal. No output shall be produced if the files are identical.

OPTIONS

The *diff* utility shall conform to XBD [Section 12.2](#) (on page 215).

The following options shall be supported:

- b** Cause any amount of white space at the end of a line to be treated as a single <newline> (that is, the white-space characters preceding the <newline> are ignored) and other strings of white-space characters, not including <newline> characters, to compare equal.
 - c** Produce output in a form that provides three lines of copied context.
 - C *n*** Produce output in a form that provides *n* lines of copied context (where *n* shall be interpreted as a positive decimal integer).
 - e** Produce output in a form suitable as input for the *ed* utility, which can then be used to convert *file1* into *file2*.
 - f** Produce output in an alternative form, similar in format to **-e**, but not intended to be suitable as input for the *ed* utility, and in the opposite order.
 - r** Apply *diff* recursively to files and directories of the same name when *file1* and *file2* are both directories.
- The *diff* utility shall detect infinite loops; that is, entering a previously visited directory that is an ancestor of the last file encountered. When it detects an infinite loop, *diff* shall write a diagnostic message to standard error and shall either recover its position in the hierarchy or terminate.
- u** Produce output in a form that provides three lines of unified context.
 - U *n*** Produce output in a form that provides *n* lines of unified context (where *n* shall be interpreted as a non-negative decimal integer).

OPERANDS

The following operands shall be supported:

file1, file2 A pathname of a file to be compared. If either the *file1* or *file2* operand is '-', the standard input shall be used in its place.

If both *file1* and *file2* are directories, *diff* shall not compare block special files, character special files, or FIFO special files to any files and shall not compare regular files to directories. Further details are as specified in [Diff Directory Comparison Format](#) (on page 2600). The behavior of *diff* on other file types is implementation-defined when found in directories.

If only one of *file1* and *file2* is a directory, *diff* shall be applied to the non-directory file and the file contained in the directory file with a filename that is the same as the last component of the non-directory file.

diff*Utilities*83749 **STDIN**

83750 The standard input shall be used only if one of the *file1* or *file2* operands references standard
 83751 input. See the INPUT FILES section.

83752 **INPUT FILES**

83753 The input files may be of any type.

83754 **ENVIRONMENT VARIABLES**

83755 The following environment variables shall affect the execution of *diff*:

83756 *LANG* Provide a default value for the internationalization variables that are unset or null.
 83757 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 83758 variables used to determine the values of locale categories.)

83759 *LC_ALL* If set to a non-empty string value, override the values of all the other
 83760 internationalization variables.

83761 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 83762 characters (for example, single-byte as opposed to multi-byte characters in
 83763 arguments and input files).

83764 *LC_MESSAGES*

83765 Determine the locale that should be used to affect the format and contents of
 83766 diagnostic messages written to standard error and informative messages written to
 83767 standard output.

83768 *LC_TIME* Determine the locale for affecting the format of file timestamps written with the *-C*
 83769 and *-c* options.

83770 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

83771 *TZ* Determine the timezone used for calculating file timestamps written with a context
 83772 format. If *TZ* is unset or null, an unspecified default timezone shall be used.

83773 **ASYNCHRONOUS EVENTS**

83774 Default.

83775 **STDOUT**83776 **Diff Directory Comparison Format**

83777 If both *file1* and *file2* are directories, the following output formats shall be used.

83778 In the POSIX locale, each file that is present in only one directory shall be reported using the
 83779 following format:

83780 "Only in %s: %s\n", <directory pathname>, <filename>

83781 In the POSIX locale, subdirectories that are common to the two directories may be reported with
 83782 the following format:

83783 "Common subdirectories: %s and %s\n", <directory1 pathname>,
 83784 <directory2 pathname>

83785 For each file common to the two directories, if the two files are not to be compared: if the two
 83786 files have the same device ID and file serial number, or are both block special files that refer to
 83787 the same device, or are both character special files that refer to the same device, in the POSIX
 83788 locale the output format is unspecified. Otherwise, in the POSIX locale an unspecified format
 83789 shall be used that contains the pathnames of the two files.

83790 For each file common to the two directories, if the files are compared and are identical, no

output shall be written. If the two files differ, the following format is written:

```
"diff %s %s %s\n", <diff_options>, <filename1>, <filename2>
```

where <diff_options> are the options as specified on the command line.

All directory pathnames listed in this section shall be relative to the original command line arguments. All other names of files listed in this section shall be filenames (pathname components).

Diff Binary Output Format

In the POSIX locale, if one or both of the files being compared are not text files, it is implementation-defined whether *diff* uses the binary file output format or the other formats as specified below. The binary file output format shall contain the pathnames of two files being compared and the string "differ".

If both files being compared are text files, depending on the options specified, one of the following formats shall be used to write the differences.

Diff Default Output Format

The default (without *-e*, *-f*, *-c*, *-C*, *-u*, or *-U* options) *diff* utility output shall contain lines of these forms:

```
"%da%d\n", <num1>, <num2>
```

```
"%da%d,%d\n", <num1>, <num2>, <num3>
```

```
"%dd%d\n", <num1>, <num2>
```

```
"%d,%dd%d\n", <num1>, <num2>, <num3>
```

```
"%dc%d\n", <num1>, <num2>
```

```
"%d,%dc%d\n", <num1>, <num2>, <num3>
```

```
"%dc%d,%d\n", <num1>, <num2>, <num3>
```

```
"%d,%dc%d,%d\n", <num1>, <num2>, <num3>, <num4>
```

These lines resemble *ed* subcommands to convert *file1* into *file2*. The line numbers before the action letters shall pertain to *file1*; those after shall pertain to *file2*. Thus, by exchanging *a* for *d* and reading the line in reverse order, one can also determine how to convert *file2* into *file1*. As in *ed*, identical pairs (where *num1*=*num2*) are abbreviated as a single number.

Following each of these lines, *diff* shall write to standard output all lines affected in the first file using the format:

<A%s", <line>

and all lines affected in the second file using the format:

>A%s", <line>

If there are lines affected in both *file1* and *file2* (as with the *c* subcommand), the changes are separated with a line consisting of three <hyphen> characters:

```
"---\n"
```

diff*Utilities***Diff -e Output Format**

With the **-e** option, a script shall be produced that shall, when provided as input to *ed*, along with an appended **w** (write) command, convert *file1* into *file2*. Only the **a** (append), **c** (change), **d** (delete), **i** (insert), and **s** (substitute) commands of *ed* shall be used in this script. Text lines, except those consisting of the single character <period> (**.**), shall be output as they appear in the file.

Diff -f Output Format

With the **-f** option, an alternative format of script shall be produced. It is similar to that produced by **-e**, with the following differences:

1. It is expressed in reverse sequence; the output of **-e** orders changes from the end of the file to the beginning; the **-f** from beginning to end.
2. The command form **<lines> <command-letter>** used by **-e** is reversed. For example, **10c** with **-e** would be **c10** with **-f**.
3. The form used for ranges of line numbers is **<space>-separated**, rather than **<comma>-separated**.

Diff -c or -C Output Format

With the **-c** or **-C** option, the output format shall consist of affected lines along with surrounding lines of context. The affected lines shall show which ones need to be deleted or changed in *file1*, and those added from *file2*. With the **-c** option, three lines of context, if available, shall be written before and after the affected lines. With the **-C** option, the user can specify how many lines of context are written. The exact format follows.

The name and last modification time of each file shall be output in the following format:

```
**** %s %s\n", file1, <file1 timestamp>
--- %s %s\n", file2, <file2 timestamp>
```

Each **<file>** field shall be the pathname of the corresponding file being compared. The pathname written for standard input is unspecified.

In the POSIX locale, each **<timestamp>** field shall be equivalent to the output from the following command:

```
date "+%a %b %e %T %Y"
```

without the trailing **<newline>**, executed at the time of last modification of the corresponding file (or the current time, if the file is standard input).

Then, the following output formats shall be applied for every set of changes.

First, a line shall be written in the following format:

```
*****\n"
```

Next, the range of lines in *file1* shall be written in the following format if the range contains two or more lines:

```
**** %d,%d ****\n", <beginning line number>, <ending line number>
```

and the following format otherwise:

```
**** %d ****\n", <ending line number>
```

The ending line number of an empty range shall be the number of the preceding line, or 0 if the

range is at the start of the file.

Next, the affected lines along with lines of context (unaffected lines) shall be written. Unaffected lines shall be written in the following format:

"ΔΔ%s", <unaffected_line>

Deleted lines shall be written as:

"-Δ%s", <deleted_line>

Changed lines shall be written as:

"!Δ%s", <changed_line>

Next, the range of lines in *file2* shall be written in the following format if the range contains two or more lines:

"--- %d,%d ----\n", <beginning line number>, <ending line number>

and the following format otherwise:

"--- %d ----\n", <ending line number>

Then, lines of context and changed lines shall be written as described in the previous formats. Lines added from *file2* shall be written in the following format:

"+Δ%s", <added_line>

Diff -u or -U Output Format

The -u or -U options behave like the -c or -C options, except that the context lines are not repeated; instead, the context, deleted, and added lines are shown together, interleaved. The exact format follows.

The name and last modification time of each file shall be output in the following format:

"---Δ%s%sΔ%s0, file1, <file1 timestamp>, <file1 frac>, <file1 zone>

"+++Δ%s%sΔ%s0, file2, <file2 timestamp>, <file2 frac>, <file2 zone>

Each <file> field shall be the pathname of the corresponding file being compared, or the single character '-' if standard input is being compared. However, if the pathname contains a <tab> or a <newline>, or if it does not consist entirely of characters taken from the portable character set, the behavior is implementation-defined.

Each <timestamp> field shall be equivalent to the output from the following command:

date '+%Y-%m-%dΔ%H:%M:%S'

without the trailing <newline>, executed at the time of last modification of the corresponding file (or the current time, if the file is standard input).

Each <frac> field shall be either empty, or a decimal point followed by at least one decimal digit, indicating the fractional-seconds part (if any) of the file timestamp. The number of fractional digits shall be at least the number needed to represent the file's timestamp without loss of information.

Each <zone> field shall be of the form "shhmm", where "shh" is a signed two-digit decimal number in the range -24 through +25, and "mm" is an unsigned two-digit decimal number in the range 00 through 59. It represents the timezone of the timestamp as the number of hours (hh) and minutes (mm) east (+) or west (-) of UTC for the timestamp. If the hours and minutes are both zero, the sign shall be '+'. However, if the timezone is not an integral number of minutes

diff*Utilities*

83907 away from UTC, the *<zone>* field is implementation-defined.

83908 Then, the following output formats shall be applied for every set of changes.

83909 First, the range of lines in each file shall be written in the following format:

83910 "@@Δ-%sΔ+%sΔ@@", *<file1 range>*, *<file2 range>*

83911 Each *<range>* field shall be of the form:

83912 "%ld", *<beginning line number>*

83913 if the range contains exactly one line, and:

83914 "%ld,%ld", *<beginning line number>*, *<number of lines>*

83915 otherwise. If a range is empty, its beginning line number shall be the number of the line just

83916 before the range, or 0 if the empty range starts the file.

83917 Next, the affected lines along with lines of context shall be written. Each non-empty unaffected

83918 line shall be written in the following format:

83919 "Δ%s", *<unaffected_line>*

83920 where the contents of the unaffected line shall be taken from *file1*. It is implementation-defined

83921 whether an empty unaffected line is written as an empty line or a line containing a single

83922 *<space>* character. This line also represents the same line of *file2*, even though *file2*'s line may

83923 contain different contents due to the *-b*. Deleted lines shall be written as:

83924 "-%s", *<deleted_line>*

83925 Added lines shall be written as:

83926 "+%s", *<added_line>*

83927 The order of lines written shall be the same as that of the corresponding file. A deleted line shall

83928 never be written immediately after an added line.

83929 If *-U n* is specified, the output shall contain no more than *n* consecutive unaffected lines; and if

83930 the output contains an affected line and this line is adjacent to up to *n* consecutive unaffected

83931 lines in the corresponding file, the output shall contain these unaffected lines. *-u* shall act like

83932 *-U3*.

83933 **STDERR**

83934 The standard error shall be used only for diagnostic messages.

83935 **OUTPUT FILES**

83936 None.

83937 **EXTENDED DESCRIPTION**

83938 None.

83939 **EXIT STATUS**

83940 The following exit values shall be returned:

83941 0 No differences were found.

83942 1 Differences were found.

83943 >1 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

If lines at the end of a file are changed and other lines are added, *diff* output may show this as a delete and add, as a change, or as a change and add; *diff* is not expected to know which happened and users should not care about the difference in output as long as it clearly shows the differences between the files.

EXAMPLES

If **dir1** is a directory containing a directory named **x**, **dir2** is a directory containing a directory named **x**, **dir1/x** and **dir2/x** both contain files named **date.out**, and **dir2/x** contains a file named **y**, the command:

```
diff -r dir1 dir2
```

could produce output similar to:

```
Common subdirectories: dir1/x and dir2/x
Only in dir2/x: y
diff -r dir1/x/date.out dir2/x/date.out
1c1
< Mon Jul  2 13:12:16 PDT 1990
---
> Tue Jun 19 21:41:39 PDT 1990
```

RATIONALE

The **-h** option was omitted because it was insufficiently specified and does not add to applications portability.

Historical implementations employ algorithms that do not always produce a minimum list of differences; the current language about making every effort is the best this volume of POSIX.1-2008 can do, as there is no metric that could be employed to judge the quality of implementations against any and all file contents. The statement “This list should be minimal” clearly implies that implementations are not expected to provide the following output when comparing two 100-line files that differ in only one character on a single line:

```
1,100c1,100
all 100 lines from file1 preceded with "< "
---
all 100 lines from file2 preceded with "> "
```

The “Only in” messages required when the **-r** option is specified are not used by most historical implementations if the **-e** option is also specified. It is required here because it provides useful information that must be provided to update a target directory hierarchy to match a source hierarchy. The “Common subdirectories” messages are written by System V and 4.3 BSD when the **-r** option is specified. They are allowed here but are not required because they are reporting on something that is the same, not reporting a difference, and are not needed to update a target hierarchy.

The **-c** option, which writes output in a format using lines of context, has been included. The format is useful for a variety of reasons, among them being much improved readability and the ability to understand difference changes when the target file has line numbers that differ from another similar, but slightly different, copy. The *patch* utility is most valuable when working with difference listings using a context format. The BSD version of **-c** takes an optional argument specifying the amount of context. Rather than overloading **-c** and breaking the Utility Syntax Guidelines for *diff*, the standard developers decided to add a separate option for

diff*Utilities*

specifying a context diff with a specified amount of context (**-C**). Also, the format for context *diffs* was extended slightly in 4.3 BSD to allow multiple changes that are within context lines from each other to be merged together. The output format contains an additional four *<asterisk>* characters after the range of affected lines in the first filename. This was to provide a flag for old programs (like old versions of *patch*) that only understand the old context format. The version of context described here does not require that multiple changes within context lines be merged, but it does not prohibit it either. The extension is upwards-compatible, so any vendors that wish to retain the old version of *diff* can do so by adding the extra four *<asterisk>* characters (that is, utilities that currently use *diff* and understand the new merged format will also understand the old unmerged format, but not *vice versa*).

The **-u** and **-U** options of GNU *diff* have been included. Their output format, designed by Wayne Davison, takes up less space than **-c** and **-C** format, and in many cases is easier to read. The format's timestamps do not vary by locale, so *LC_TIME* does not affect it. The format's line numbers are rendered with the **%ld** format, not **%d**, because the file format notation rules would allow extra *<blank>* characters to appear around the numbers.

The substitute command was added as an additional format for the **-e** option. This was added to provide implementations with a way to fix the classic "dot alone on a line" bug present in many versions of *diff*. Since many implementations have fixed this bug, the standard developers decided not to standardize broken behavior, but rather to provide the necessary tool for fixing the bug. One way to fix this bug is to output two periods whenever a lone period is needed, then terminate the append command with a period, and then use the substitute command to convert the two periods into one period.

The BSD-derived **-r** option was added to provide a mechanism for using *diff* to compare two file system trees. This behavior is useful, is standard practice on all BSD-derived systems, and is not easily reproducible with the *find* utility.

The requirement that *diff* not compare files in some circumstances, even though they have the same name, is based on the actual output of historical implementations. The specified behavior precludes the problems arising from running into FIFOs and other files that would cause *diff* to hang waiting for input with no indication to the user that *diff* was hung. An earlier version of this standard specified the output format more precisely, but in practice this requirement was widely ignored and the benefit of standardization seemed small, so it is now unspecified. In most common usage, *diff -r* should indicate differences in the file hierarchies, not the difference of contents of devices pointed to by the hierarchies.

Many early implementations of *diff* require seekable files. Since the System Interfaces volume of POSIX.1-2008 supports named pipes, the standard developers decided that such a restriction was unreasonable. Note also that the allowed filename – almost always refers to a pipe.

No directory search order is specified for *diff*. The historical ordering is, in fact, not optimal, in that it prints out all of the differences at the current level, including the statements about all common subdirectories before recursing into those subdirectories.

The message:

```
"diff %s %s %s\n", <diff_options>, <filename1>, <filename2>
```

does not vary by locale because it is the representation of a command, not an English sentence.

FUTURE DIRECTIONS

None.

SEE ALSO*cmp, comm, ed, find*

XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The `-f` option is added.

The output format for `-c` or `-C` format is changed to align with changes to the IEEE P1003.2b draft standard resulting from IEEE PASC Interpretation 1003.2 #71.

The normative text is reworded to avoid use of the term “must” for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/20 is applied, changing the STDOUT section. This changes the specification of *diff* `-c` so that it agrees with existing practice when contexts contain zero lines or one line.**Issue 7**

Austin Group Interpretations 1003.1-2001 #115 and #114 are applied.

Austin Group Interpretation 1003.1-2001 #192 is applied, clarifying the behavior if both files are non-text files.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the `-u` option.

dirname

Utilities

84058 **NAME**

84059 dirname — return the directory portion of a pathname

84060 **SYNOPSIS**84061 dirname *string*84062 **DESCRIPTION**

84063 The *string* operand shall be treated as a pathname, as defined in XBD [Section 3.266](#) (on page 75).
 84064 The string *string* shall be converted to the name of the directory containing the filename
 84065 corresponding to the last pathname component in *string*, performing actions equivalent to the
 84066 following steps in order:

- 84067 1. If *string* is *//*, skip steps 2 to 5.
- 84068 2. If *string* consists entirely of <slash> characters, *string* shall be set to a single <slash>
 84069 character. In this case, skip steps 3 to 8.
- 84070 3. If there are any trailing <slash> characters in *string*, they shall be removed.
- 84071 4. If there are no <slash> characters remaining in *string*, *string* shall be set to a single
 84072 <period> character. In this case, skip steps 5 to 8.
- 84073 5. If there are any trailing non-<slash> characters in *string*, they shall be removed.
- 84074 6. If the remaining *string* is *//*, it is implementation-defined whether steps 7 and 8 are
 84075 skipped or processed.
- 84076 7. If there are any trailing <slash> characters in *string*, they shall be removed.
- 84077 8. If the remaining *string* is empty, *string* shall be set to a single <slash> character.

84078 The resulting string shall be written to standard output.

84079 **OPTIONS**

84080 None.

84081 **OPERANDS**

84082 The following operand shall be supported:

84083 *string* A string.84084 **STDIN**

84085 Not used.

84086 **INPUT FILES**

84087 None.

84088 **ENVIRONMENT VARIABLES**84089 The following environment variables shall affect the execution of *dirname*:

84090 84091 84092	LANG	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
84093 84094	LC_ALL	If set to a non-empty string value, override the values of all the other internationalization variables.
84095 84096 84097	LC_CTYPE	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

Utilities

dirname

84098 **LC_MESSAGES**

84099 Determine the locale that should be used to affect the format and contents of
84100 diagnostic messages written to standard error.

84101 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

84102 **ASYNCHRONOUS EVENTS**

84103 Default.

84104 **STDOUT**

84105 The *dirname* utility shall write a line to the standard output in the following format:

84106 "%s\n", <resulting string>

84107 **STDERR**

84108 The standard error shall be used only for diagnostic messages.

84109 **OUTPUT FILES**

84110 None.

84111 **EXTENDED DESCRIPTION**

84112 None.

84113 **EXIT STATUS**

84114 The following exit values shall be returned:

84115 0 Successful completion.

84116 >0 An error occurred.

84117 **CONSEQUENCES OF ERRORS**

84118 Default.

84119 **APPLICATION USAGE**

84120 The definition of *pathname* specifies implementation-defined behavior for pathnames starting
84121 with two <slash> characters. Therefore, applications shall not arbitrarily add <slash> characters
84122 to the beginning of a pathname unless they can ensure that there are more or less than two or are
84123 prepared to deal with the implementation-defined consequences.

84124 **EXAMPLES**

84125

84126

84127

84128

84129

84130

84131

84132

84133

84134

84135

Command	Results
<i>dirname</i> /	/
<i>dirname</i> //	/ or //
<i>dirname</i> /a/b/	/a
<i>dirname</i> //a//b//	//a
<i>dirname</i>	Unspecified
<i>dirname</i> a	.(\$? = 0)
<i>dirname</i> ""	.(\$? = 0)
<i>dirname</i> /a	/
<i>dirname</i> /a/b	/a
<i>dirname</i> a/b	a

84136 **RATIONALE**

84137 The *dirname* utility originated in System III. It has evolved through the System V releases to a
84138 version that matches the requirements specified in this description in System V Release 3. 4.3
84139 BSD and earlier versions did not include *dirname*.

84140 The behaviors of *basename* and *dirname* in this volume of POSIX.1-2008 have been coordinated so

dirname

Utilities

84141 that when *string* is a valid pathname:

84142 `$(basename "string")`

84143 would be a valid filename for the file in the directory:

84144 `$(dirname "string")`

84145 This would not work for the versions of these utilities in early proposals due to the way
84146 processing of trailing <slash> characters was specified. Consideration was given to leaving
84147 processing unspecified if there were trailing <slash> characters, but this cannot be done; XBD
84148 [Section 3.266](#) (on page 75) allows trailing <slash> characters. The *basename* and *dirname* utilities
84149 have to specify consistent handling for all valid pathnames.

84150 **FUTURE DIRECTIONS**

84151 None.

84152 **SEE ALSO**

84153 [Section 2.5](#) (on page 2301), *basename*

84154 XBD [Section 3.266](#) (on page 75), [Chapter 8](#) (on page 173)

84155 **CHANGE HISTORY**

84156 First released in Issue 2.

Utilities

du**NAME**

du — estimate file space usage

SYNOPSIS

du [**-a** | **-s**] [**-kx**] [**-H** | **-L**] [*file*...]

DESCRIPTION

By default, the *du* utility shall write to standard output the size of the file space allocated to, and the size of the file space allocated to each subdirectory of, the file hierarchy rooted in each of the specified files. By default, when a symbolic link is encountered on the command line or in the file hierarchy, *du* shall count the size of the symbolic link (rather than the file referenced by the link), and shall not follow the link to another portion of the file hierarchy. The size of the file space allocated to a file of type directory shall be defined as the sum total of space allocated to all files in the file hierarchy rooted in the directory plus the space allocated to the directory itself.

When *du* cannot *stat()* files or *stat()* or read directories, it shall report an error condition and the final exit status is affected. Files with multiple links shall be counted and written for only one entry. The directory entry that is selected in the report is unspecified. By default, file sizes shall be written in 512-byte units, rounded up to the next 512-byte unit.

OPTIONS

The *du* utility shall conform to XBD [Section 12.2](#) (on page 215).

The following options shall be supported:

- a** In addition to the default output, report the size of each file not of type directory in the file hierarchy rooted in the specified file. Regardless of the presence of the **-a** option, non-directories given as *file* operands shall always be listed.
- H** If a symbolic link is specified on the command line, *du* shall count the size of the file or file hierarchy referenced by the link.
- k** Write the files sizes in units of 1 024 bytes, rather than the default 512-byte units.
- L** If a symbolic link is specified on the command line or encountered during the traversal of a file hierarchy, *du* shall count the size of the file or file hierarchy referenced by the link.
- s** Instead of the default output, report only the total sum for each of the specified files.
- x** When evaluating file sizes, evaluate only those files that have the same device as the file specified by the *file* operand.

Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered an error. The last option specified shall determine the behavior of the utility.

OPERANDS

The following operand shall be supported:

- file* The pathname of a file whose size is to be written. If no *file* is specified, the current directory shall be used.

STDIN

Not used.

INPUT FILES

None.

84199 **ENVIRONMENT VARIABLES**84200 The following environment variables shall affect the execution of *du*:

84201 *LANG* Provide a default value for the internationalization variables that are unset or null.
 84202 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 84203 variables used to determine the values of locale categories.)

84204 *LC_ALL* If set to a non-empty string value, override the values of all the other
 84205 internationalization variables.

84206 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 84207 characters (for example, single-byte as opposed to multi-byte characters in
 84208 arguments).

84209 *LC_MESSAGES*

84210 Determine the locale that should be used to affect the format and contents of
 84211 diagnostic messages written to standard error.

84212 *XSI NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

84213 **ASYNCHRONOUS EVENTS**

84214 Default.

84215 **STDOUT**

84216 The output from *du* shall consist of the amount of space allocated to a file and the name of the
 84217 file, in the following format:

84218 "%d %s\n", <size>, <pathname>

84219 **STDERR**

84220 The standard error shall be used only for diagnostic messages.

84221 **OUTPUT FILES**

84222 None.

84223 **EXTENDED DESCRIPTION**

84224 None.

84225 **EXIT STATUS**

84226 The following exit values shall be returned:

84227 0 Successful completion.

84228 >0 An error occurred.

84229 **CONSEQUENCES OF ERRORS**

84230 Default.

84231 **APPLICATION USAGE**

84232 None.

84233 **EXAMPLES**

84234 None.

84235 **RATIONALE**

84236 The use of 512-byte units is historical practice and maintains compatibility with *ls* and other
 84237 utilities in this volume of POSIX.1-2008. This does not mandate that the file system itself be
 84238 based on 512-byte blocks. The *-k* option was added as a compromise measure. It was agreed by
 84239 the standard developers that 512 bytes was the best default unit because of its complete
 84240 historical consistency on System V (*versus* the mixed 512/1 024-byte usage on BSD systems), and
 84241 that a *-k* option to switch to 1024-byte units was a good compromise. Users who prefer the

1 024-byte quantity can easily alias *du* to *du -k* without breaking the many historical scripts relying on the 512-byte units.

The **-b** option was added to an early proposal to provide a resolution to the situation where System V and BSD systems give figures for file sizes in *blocks*, which is an implementation-defined concept. (In common usage, the block size is 512 bytes for System V and 1 024 bytes for BSD systems.) However, **-b** was later deleted, since the default was eventually decided as 512-byte units.

Historical file systems provided no way to obtain exact figures for the space allocation given to files. There are two known areas of inaccuracies in historical file systems: cases of *indirect blocks* being used by the file system or *sparse* files yielding incorrectly high values. An indirect block is space used by the file system in the storage of the file, but that need not be counted in the space allocated to the file. A *sparse* file is one in which an *lseek()* call has been made to a position beyond the end of the file and data has subsequently been written at that point. A file system need not allocate all the intervening zero-filled blocks to such a file. It is up to the implementation to define exactly how accurate its methods are.

The **-a** and **-s** options were mutually-exclusive in the original version of *du*. The POSIX Shell and Utilities description is implied by the language in the SVID where **-s** is described as causing “only the grand total” to be reported. Some systems may produce output for **-sa**, but a Strictly Conforming POSIX Shell and Utilities Application cannot use that combination.

The **-a** and **-s** options were adopted from the SVID except that the System V behavior of not listing non-directories explicitly given as operands, unless the **-a** option is specified, was considered a bug; the BSD-based behavior (report for all operands) is mandated. The default behavior of *du* in the SVID with regard to reporting the failure to read files (it produces no messages) was considered counter-intuitive, and thus it was specified that the POSIX Shell and Utilities default behavior shall be to produce such messages. These messages can be turned off with shell redirection to achieve the System V behavior.

The **-x** option is historical practice on recent BSD systems. It has been adopted by this volume of POSIX.1-2008 because there was no other historical method of limiting the *du* search to a single file hierarchy. This limitation of the search is necessary to make it possible to obtain file space usage information about a file system on which other file systems are mounted, without having to resort to a lengthy *find* and *awk* script.

FUTURE DIRECTIONS

None.

SEE ALSO

ls

XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

XSH *fstatat()*

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The APPLICATION USAGE section is added.

The obsolescent **-r** option is removed.

The Open Group Corrigendum U025/3 is applied. The *du* utility is reinstated, as it had incorrectly been marked LEGACY in Issue 5.

du*Utilities*

- 84287 The **-H** and **-L** options for symbolic links are added as described in the IEEE P1003.2b draft
84288 standard.
- 84289 **Issue 7**
84290 The *du* utility is moved from the User Portability Utilities option to the Base. User Portability
84291 Utilities is now an option for interactive utilities.
- 84292 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

echo

NAME

echo — write arguments to standard output

SYNOPSIS

echo [*string*...]

DESCRIPTION

The *echo* utility writes its arguments to standard output, followed by a <newline>. If there are no arguments, only the <newline> is written.

OPTIONS

The *echo* utility shall not recognize the "--" argument in the manner specified by Guideline 10 of XBD Section 12.2 (on page 215); "--" shall be recognized as a string operand.

Implementations shall not support any options.

OPERANDS

The following operands shall be supported:

string A string to be written to standard output. If the first operand is -n, or if any of the operands contain a <backslash> character, the results are implementation-defined.

XSI

On XSI-conformant systems, if the first operand is -n, it shall be treated as a string, not an option. The following character sequences shall be recognized on XSI-conformant systems within any of the arguments:

\a Write an <alert>.

\b Write a <backspace>.

\c Suppress the <newline> that otherwise follows the final argument in the output. All characters following the '\c' in the arguments shall be ignored.

\f Write a <form-feed>.

\n Write a <newline>.

\r Write a <carriage-return>.

\t Write a <tab>.

\v Write a <vertical-tab>.

\l Write a <backslash> character.

\0*num* Write an 8-bit value that is the zero, one, two, or three-digit octal number *num*.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *echo*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

echo

Utilities

84333	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
84334		
84335	XSI <i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
84336		
84337		
84338	<i>LC_MESSAGES</i>	
84339		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
84340		
84341	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
84342	ASYNCHRONOUS EVENTS	
84343		Default.
84344	STDOUT	
84345		The <i>echo</i> utility arguments shall be separated by single <space> characters and a <newline> character shall follow the last argument. Output transformations shall occur based on the escape sequences in the input. See the OPERANDS section.
84346	XSI	
84347		
84348	STDERR	
84349		The standard error shall be used only for diagnostic messages.
84350	OUTPUT FILES	
84351		None.
84352	EXTENDED DESCRIPTION	
84353		None.
84354	EXIT STATUS	
84355		The following exit values shall be returned:
84356	0	Successful completion.
84357	>0	An error occurred.
84358	CONSEQUENCES OF ERRORS	
84359		Default.
84360	APPLICATION USAGE	
84361		It is not possible to use <i>echo</i> portably across all POSIX systems unless both <i>-n</i> (as the first argument) and escape sequences are omitted.
84362		
84363		The <i>printf</i> utility can be used portably to emulate any of the traditional behaviors of the <i>echo</i> utility as follows (assuming that <i>IFS</i> has its standard value or is unset):
84364		
84365		• The historic System V <i>echo</i> and the requirements on XSI implementations in this volume of POSIX.1-2008 are equivalent to:
84366		
84367		<code>printf "%b\n" "\$*"</code>
84368		• The BSD <i>echo</i> is equivalent to:
84369		<code>if ["X\$1" = "X-n"]</code>
84370		<code>then</code>
84371		<code> shift</code>
84372		<code> printf "%s" "\$*"</code>
84373		<code>else</code>
84374		<code> printf "%s\n" "\$*"</code>
84375		<code>fi</code>

84376 New applications are encouraged to use *printf* instead of *echo*.

84377 EXAMPLES

84378 None.

84379 RATIONALE

84380 The *echo* utility has not been made obsolescent because of its extremely widespread use in
 84381 historical applications. Conforming applications that wish to do prompting without <newline>
 84382 characters or that could possibly be expecting to echo a **-n**, should use the *printf* utility derived
 84383 from the Ninth Edition system.

84384 As specified, *echo* writes its arguments in the simplest of ways. The two different historical
 84385 versions of *echo* vary in fatally incompatible ways.

84386 The BSD *echo* checks the first argument for the string **-n** which causes it to suppress the
 84387 <newline> that would otherwise follow the final argument in the output.

84388 The System V *echo* does not support any options, but allows escape sequences within its
 84389 operands, as described for XSI implementations in the OPERANDS section.

84390 The *echo* utility does not support Utility Syntax Guideline 10 because historical applications
 84391 depend on *echo* to echo *all* of its arguments, except for the **-n** option in the BSD version.

84392 FUTURE DIRECTIONS

84393 None.

84394 SEE ALSO

84395 *printf*

84396 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

84397 CHANGE HISTORY

84398 First released in Issue 2.

84399 Issue 5

84400 In the OPTIONS section, the last sentence is changed to indicate that implementations “do not”
 84401 support any options; in the previous issue this said “need not”.

84402 Issue 6

84403 The following new requirements on POSIX implementations derive from alignment with the
 84404 Single UNIX Specification:

- 84405 • A set of character sequences is defined as *string* operands.
- 84406 • *LC_CTYPE* is added to the list of environment variables affecting *echo*.
- 84407 • In the OPTIONS section, implementations shall not support any options.

84408 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/21 is applied, so that the *echo* utility can
 84409 accommodate historical BSD behavior.

84410 Issue 7

84411 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

ed*Utilities*84412 **NAME**

84413 ed — edit text

84414 **SYNOPSIS**84415 ed [-p *string*] [-s] [*file*]84416 **DESCRIPTION**

84417 The *ed* utility is a line-oriented text editor that uses two modes: *command mode* and *input mode*. In
 84418 command mode the input characters shall be interpreted as commands, and in input mode they
 84419 shall be interpreted as text. See the EXTENDED DESCRIPTION section.

84420 If an operand is '-' , the results are unspecified.

84421 **OPTIONS**

84422 The *ed* utility shall conform to XBD Section 12.2 (on page 215), except for the unspecified usage
 84423 of '-' .

84424 The following options shall be supported:

- 84425 -p *string* Use *string* as the prompt string when in command mode. By default, there shall be
 84426 no prompt string.
- 84427 -s Suppress the writing of byte counts by **e**, **E**, **r**, and **w** commands and of the '!'
 84428 prompt after a *!command*.

84429 **OPERANDS**

84430 The following operand shall be supported:

- 84431 *file* If the *file* argument is given, *ed* shall simulate an **e** command on the file named by
 84432 the pathname, *file*, before accepting commands from the standard input.

84433 **STDIN**

84434 The standard input shall be a text file consisting of commands, as described in the EXTENDED
 84435 DESCRIPTION section.

84436 **INPUT FILES**

84437 The input files shall be text files.

84438 **ENVIRONMENT VARIABLES**

84439 The following environment variables shall affect the execution of *ed*:

- 84440 HOME Determine the pathname of the user's home directory.
- 84441 LANG Provide a default value for the internationalization variables that are unset or null.
 84442 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 84443 variables used to determine the values of locale categories.)
- 84444 LC_ALL If set to a non-empty string value, override the values of all the other
 84445 internationalization variables.
- 84446 LC_COLLATE Determine the locale for the behavior of ranges, equivalence classes, and multi-
 84447 character collating elements within regular expressions.
- 84449 LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as
 84450 characters (for example, single-byte as opposed to multi-byte characters in
 84451 arguments and input files) and the behavior of character classes within regular
 84452 expressions.

84453 *LC_MESSAGES*

84454 Determine the locale that should be used to affect the format and contents of
 84455 diagnostic messages written to standard error and informative messages written to
 84456 standard output.

84457 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

84458 **ASYNCHRONOUS EVENTS**

84459 The *ed* utility shall take the standard action for all signals (see the ASYNCHRONOUS EVENTS
 84460 section in [Section 1.4](#), on page 2288) with the following exceptions:

84461 SIGINT The *ed* utility shall interrupt its current activity, write the string "?\n" to standard
 84462 output, and return to command mode (see the EXTENDED DESCRIPTION
 84463 section).

84464 SIGHUP If the buffer is not empty and has changed since the last write, the *ed* utility shall
 84465 attempt to write a copy of the buffer in a file. First, the file named **ed.hup** in the
 84466 current directory shall be used; if that fails, the file named **ed.hup** in the directory
 84467 named by the *HOME* environment variable shall be used. In any case, the *ed* utility
 84468 shall exit without writing the file to the currently remembered pathname and
 84469 without returning to command mode.

84470 SIGQUIT The *ed* utility shall ignore this event.

84471 **STDOUT**

84472 Various editing commands and the prompting feature (see **-p**) write to standard output, as
 84473 described in the EXTENDED DESCRIPTION section.

84474 **STDERR**

84475 The standard error shall be used only for diagnostic messages.

84476 **OUTPUT FILES**

84477 The output files shall be text files whose formats are dependent on the editing commands given.

84478 **EXTENDED DESCRIPTION**

84479 The *ed* utility shall operate on a copy of the file it is editing; changes made to the copy shall have
 84480 no effect on the file until a **w** (write) command is given. The copy of the text is called the *buffer*.

84481 Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a
 84482 single-character *command*, possibly followed by parameters to that command. These addresses
 84483 specify one or more lines in the buffer. Every command that requires addresses has default
 84484 addresses, so that the addresses very often can be omitted. If the **-p** option is specified, the
 84485 prompt string shall be written to standard output before each command is read.

84486 In general, only one command can appear on a line. Certain commands allow text to be input.
 84487 This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be
 84488 in *input mode*. In this mode, no commands shall be recognized; all input is merely collected.
 84489 Input mode is terminated by entering a line consisting of two characters: a <period> ('.')
 84490 followed by a <newline>. This line is not considered part of the input text.

Regular Expressions in ed

The *ed* utility shall support basic regular expressions, as described in XBD [Section 9.3](#) (on page 183). Since regular expressions in *ed* are always matched against single lines (excluding the terminating <newline> characters), never against any larger section of text, there is no way for a regular expression to match a <newline>.

A null RE shall be equivalent to the last RE encountered.

Regular expressions are used in addresses to specify lines, and in some commands (for example, the *s* substitute command) to specify portions of a line to be substituted.

Addresses in ed

Addressing in *ed* relates to the current line. Generally, the current line is the last line affected by a command. The current line number is the address of the current line. If the edit buffer is not empty, the initial value for the current line shall be the last line in the edit buffer; otherwise, zero.

Addresses shall be constructed as follows:

1. The <period> character (' . ') shall address the current line.
2. The <dollar-sign> character (' \$ ') shall address the last line of the edit buffer.
3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
4. The <apostrophe>-*x* character pair (" ' x ") shall address the line marked with the mark name character *x*, which shall be a lowercase letter from the portable character set. It shall be an error if the character has not been set to mark a line or if the line that was marked is not currently present in the edit buffer.
5. A BRE enclosed by <slash> characters (' / ') shall address the first line found by searching forwards from the line following the current line toward the end of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of <slash> characters shall address the next line for which the line excluding the terminating <newline> matches the last BRE encountered. In addition, the second <slash> can be omitted at the end of a command line. Within the BRE, a <backslash>-<slash> pair (" \ / ") shall represent a literal <slash> instead of the BRE delimiter. If necessary, the search shall wrap around to the beginning of the buffer and continue up to and including the current line, so that the entire buffer is searched.
6. A BRE enclosed by <question-mark> characters (' ? ') shall address the first line found by searching backwards from the line preceding the current line toward the beginning of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of <question-mark> characters (" ? ? ") shall address the previous line for which the line excluding the terminating <newline> matches the last BRE encountered. In addition, the second <question-mark> can be omitted at the end of a command line. Within the BRE, a <backslash>-<question-mark> pair (" \ ? ") shall represent a literal <question-mark> instead of the BRE delimiter. If necessary, the search shall wrap around to the end of the buffer and continue up to and including the current line, so that the entire buffer is searched.
7. A <plus-sign> (' + ') or <hyphen> character (' - ') followed by a decimal number shall address the current line plus or minus the number. A <plus-sign> or <hyphen> character not followed by a decimal number shall address the current line plus or minus 1.

Addresses can be followed by zero or more address offsets, optionally <blank>-separated. Address offsets are constructed as follows:

- A <plus-sign> or <hyphen> character followed by a decimal number shall add or subtract, respectively, the indicated number of lines to or from the address. A <plus-sign> or <hyphen> character not followed by a decimal number shall add or subtract 1 to or from the address.
- A decimal number shall add the indicated number of lines to the address.

It shall not be an error for an intermediate address value to be less than zero or greater than the last line in the edit buffer. It shall be an error for the final address value to be less than zero or greater than the last line in the edit buffer. It shall be an error if a search for a BRE fails to find a matching line.

Commands accept zero, one, or two addresses. If more than the required number of addresses are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than the required number of addresses are provided to a command, the addresses specified first shall be evaluated and then discarded until the maximum number of valid addresses remain, for the specified command.

Addresses shall be separated from each other by a <comma> (',') or <semicolon> character (';'). In the case of a <semicolon> separator, the current line ('.') shall be set to the first address, and only then will the second address be calculated. This feature can be used to determine the starting line for forwards and backwards searches; see rules 5. and 6.

Addresses can be omitted on either side of the <comma> or <semicolon> separator, in which case the resulting address pairs shall be as follows:

Specified	Resulting
,	1 , \$
, addr	1 , addr
addr ,	addr , addr
;	. ; \$
; addr	. ; addr
addr ;	addr ; addr

Any <blank> characters included between addresses, address separators, or address offsets shall be ignored.

Commands in ed

In the following list of *ed* commands, the default addresses are shown in parentheses. The number of addresses shown in the default shall be the number expected by the command. The parentheses are not part of the address; they show that the given addresses are the default.

It is generally invalid for more than one command to appear on a line. However, any command (except **e**, **E**, **f**, **q**, **Q**, **r**, **w**, and **!**) can be suffixed by the letter **l**, **n**, or **p**; in which case, except for the **l**, **n**, and **p** commands, the command shall be executed and then the new current line shall be written as described below under the **l**, **n**, and **p** commands. When an **l**, **n**, or **p** suffix is used with an **l**, **n**, or **p** command, the command shall write to standard output as described below, but it is unspecified whether the suffix writes the current line again in the requested format or whether the suffix has no effect. For example, the **pl** command (base **p** command with an **l** suffix) shall either write just the current line or write it twice—once as specified for **p** and once as specified for **l**. Also, the **g**, **G**, **v**, and **V** commands shall take a command as a parameter.

Each address component can be preceded by zero or more <blank> characters. The command

letter can be preceded by zero or more <blank> characters. If a suffix letter (**l**, **n**, or **p**) is given, the application shall ensure that it immediately follows the command.

The **e**, **E**, **f**, **r**, and **w** commands shall take an optional *file* parameter, separated from the command letter by one or more <blank> characters.

If changes have been made in the buffer since the last **w** command that wrote the entire buffer, *ed* shall warn the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands. The *ed* utility shall write the string:

"? \n"

(followed by an explanatory message if *help mode* has been enabled via the **H** command) to standard output and shall continue in command mode with the current line number unchanged. If the **e** or **q** command is repeated with no intervening command, it shall take effect.

If a terminal disconnect (see XBD Chapter 11 (on page 199), Modem Disconnect and Closing a Device Terminal), is detected:

- If accompanied by a SIGHUP signal, the *ed* utility shall operate as described in the ASYNCHRONOUS EVENTS section for a SIGHUP signal.
- If not accompanied by a SIGHUP signal, the *ed* utility shall act as if an end-of-file had been detected on standard input.

If an end-of-file is detected on standard input:

- If the *ed* utility is in input mode, *ed* shall terminate input mode and return to command mode. It is unspecified if any partially entered lines (that is, input text without a terminating <newline>) are discarded from the input text.
- If the *ed* utility is in command mode, it shall act as if a **q** command had been entered.

If the closing delimiter of an RE or of a replacement string (for example, ' / ') in a **g**, **G**, **s**, **v**, or **V** command would be the last character before a <newline>, that delimiter can be omitted, in which case the addressed line shall be written. For example, the following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1

If an invalid command is entered, *ed* shall write the string:

"? \n"

(followed by an explanatory message if *help mode* has been enabled via the **H** command) to standard output and shall continue in command mode with the current line number unchanged.

Append Command

Synopsis: (.) a
 <text>
 .

The **a** command shall read the given text and append it after the addressed line; the current line number shall become the address of the last inserted line or, if there were none, the addressed line. Address 0 shall be valid for this command; it shall cause the appended text to be placed at the beginning of the buffer.

Change Command

Synopsis: (.,.)c
 <text>
 .

The **c** command shall delete the addressed lines, then accept input text that replaces these lines; the current line shall be set to the address of the last line input; or, if there were none, at the line after the last line deleted; if the lines deleted were originally at the end of the buffer, the current line number shall be set to the address of the new last line; if no lines remain in the buffer, the current line number shall be set to zero. Address 0 shall be valid for this command; it shall be interpreted as if address 1 were specified.

Delete Command

Synopsis: (.,.)d

The **d** command shall delete the addressed lines from the buffer. The address of the line after the last line deleted shall become the current line number; if the lines deleted were originally at the end of the buffer, the current line number shall be set to the address of the new last line; if no lines remain in the buffer, the current line number shall be set to zero.

Edit Command

Synopsis: e [*file*]

The **e** command shall delete the entire contents of the buffer and then read in the file named by the pathname *file*. The current line number shall be set to the address of the last line of the buffer. If no pathname is given, the currently remembered pathname, if any, shall be used (see the **f** command). The number of bytes read shall be written to standard output, unless the **-s** option was specified, in the following format:

"%d\n", <number of bytes read>

The name *file* shall be remembered for possible use as a default pathname in subsequent **e**, **E**, **r**, and **w** commands. If *file* is replaced by '!', the rest of the line shall be taken to be a shell command line whose output is to be read. Such a shell command line shall not be remembered as the current *file*. All marks shall be discarded upon the completion of a successful **e** command. If the buffer has changed since the last time the entire buffer was written, the user shall be warned, as described previously.

Edit Without Checking Command

Synopsis: E [*file*]

The **E** command shall possess all properties and restrictions of the **e** command except that the editor shall not check to see whether any changes have been made to the buffer since the last **w** command.

Filename Command

Synopsis: *f* [*file*]

If *file* is given, the **f** command shall change the currently remembered pathname to *file*; whether the name is changed or not, it shall then write the (possibly new) currently remembered pathname to the standard output in the following format:

"%s\n", <pathname>

The current line number shall be unchanged.

Global Command

Synopsis: (1, \$) *g*/RE/command list

In the **g** command, the first step shall be to mark every line for which the line excluding the terminating <newline> matches the given RE. Then, going sequentially from the beginning of the file to the end of the file, the given *command list* shall be executed for each marked line, with the current line number set to the address of that line. Any line modified by the *command list* shall be unmarked. When the **g** command completes, the current line number shall have the value assigned by the last command in the *command list*. If there were no matching lines, the current line number shall not be changed. A single command or the first of a list of commands shall appear on the same line as the global command. All lines of a multi-line list except the last line shall be ended with a <backslash> preceding the terminating <newline>; the **a**, **i**, and **c** commands and associated input are permitted. The **'** terminating input mode can be omitted if it would be the last line of the *command list*. An empty *command list* shall be equivalent to the **p** command. The use of the **g**, **G**, **v**, **V**, and **!** commands in the *command list* produces undefined results. Any character other than <space> or <newline> can be used instead of a <slash> to delimit the RE. Within the RE, the RE delimiter itself can be used as a literal character if it is preceded by a <backslash>.

Interactive Global Command

Synopsis: (1, \$) *G*/RE/

In the **G** command, the first step shall be to mark every line for which the line excluding the terminating <newline> matches the given RE. Then, for every such line, that line shall be written, the current line number shall be set to the address of that line, and any one command (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) shall be read and executed. A <newline> shall act as a null command (causing no action to be taken on the current line); an **'** & **'** shall cause the re-execution of the most recent non-null command executed within the current invocation of **G**. Note that the commands input as part of the execution of the **G** command can address and affect any lines in the buffer. Any line modified by the command shall be unmarked. The final value of the current line number shall be the value set by the last command successfully executed. (Note that the last command successfully executed shall be the **G** command itself if a command fails or the null command is specified.) If there were no matching lines, the current line number shall not be changed. The **G** command can be terminated by a SIGINT signal. Any character other than <space> or <newline> can be used instead of a <slash> to delimit the RE and the replacement. Within the RE, the RE delimiter itself can be used as a literal character if it is preceded by a <backslash>.

Help Command

Synopsis: h

The **h** command shall write a short message to standard output that explains the reason for the most recent ' ? ' notification. The current line number shall be unchanged.

Help-Mode Command

Synopsis: H

The **H** command shall cause *ed* to enter a mode in which help messages (see the **h** command) shall be written to standard output for all subsequent ' ? ' notifications. The **H** command alternately shall turn this mode on and off; it is initially off. If the help-mode is being turned on, the **H** command also explains the previous ' ? ' notification, if there was one. The current line number shall be unchanged.

Insert Command

Synopsis: (.) i
 <text>
 .

The **i** command shall insert the given text before the addressed line; the current line is set to the last inserted line or, if there was none, to the addressed line. This command differs from the **a** command only in the placement of the input text. Address 0 shall be valid for this command; it shall be interpreted as if address 1 were specified.

Join Command

Synopsis: (. , . +1) j

The **j** command shall join contiguous lines by removing the appropriate <newline> characters. If exactly one address is given, this command shall do nothing. If lines are joined, the current line number shall be set to the address of the joined line; otherwise, the current line number shall be unchanged.

Mark Command

Synopsis: (.) kx

The **k** command shall mark the addressed line with name *x*, which the application shall ensure is a lowercase letter from the portable character set. The address "' x" shall then refer to this line; the current line number shall be unchanged.

List Command

Synopsis: (. , .) l

The **l** command shall write to standard output the addressed lines in a visually unambiguous form. The characters listed in XBD Table 5-1 (on page 121) (' \\', '\a', '\b', '\f', '\r', '\t', '\v') shall be written as the corresponding escape sequence; the '\n' in that table is not applicable. Non-printable characters not in the table shall be written as one three-digit octal number (with a preceding <backslash> character) for each byte in the character (most significant byte first).

Long lines shall be folded, with the point of folding indicated by <newline> preceded by a <backslash>; the length at which folding occurs is unspecified, but should be appropriate for the

84737 output device. The end of each line shall be marked with a '\$', and '\$' characters within the
 84738 text shall be written with a preceding <backslash>. An **l** command can be appended to any
 84739 other command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**. The current line number shall be set to the
 84740 address of the last line written.

84741 **Move Command**

84742 *Synopsis:* (.,.)*m**address*

84743 The **m** command shall reposition the addressed lines after the line addressed by *address*.
 84744 Address 0 shall be valid for *address* and cause the addressed lines to be moved to the beginning
 84745 of the buffer. It shall be an error if *address* falls within the range of moved lines. The
 84746 current line number shall be set to the address of the last line moved.

84747 **Number Command**

84748 *Synopsis:* (.,.)*n*

84749 The **n** command shall write to standard output the addressed lines, preceding each line by its
 84750 line number and a <tab>; the current line number shall be set to the address of the last line
 84751 written. The **n** command can be appended to any command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

84752 **Print Command**

84753 *Synopsis:* (.,.)*p*

84754 The **p** command shall write to standard output the addressed lines; the current line number shall
 84755 be set to the address of the last line written. The **p** command can be appended to any command
 84756 other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

84757 **Prompt Command**

84758 *Synopsis:* *P*

84759 The **P** command shall cause *ed* to prompt with an <asterisk> ('*') (or *string*, if **-p** is specified)
 84760 for all subsequent commands. The **P** command alternatively shall turn this mode on and off; it
 84761 shall be initially on if the **-p** option is specified; otherwise, off. The current line number shall be
 84762 unchanged.

84763 **Quit Command**

84764 *Synopsis:* *q*

84765 The **q** command shall cause *ed* to exit. If the buffer has changed since the last time the entire
 84766 buffer was written, the user shall be warned, as described previously.

84767 **Quit Without Checking Command**

84768 *Synopsis:* *Q*

84769 The **Q** command shall cause *ed* to exit without checking whether changes have been made in the
 84770 buffer since the last **w** command.

Read Command

Synopsis: (\$)*r* [*file*]

The *r* command shall read in the file named by the pathname *file* and append it after the addressed line. If no *file* argument is given, the currently remembered pathname, if any, shall be used (see the *e* and *f* commands). The currently remembered pathname shall not be changed unless there is no remembered pathname. Address 0 shall be valid for *r* and shall cause the file to be read at the beginning of the buffer. If the read is successful, and *-s* was not specified, the number of bytes read shall be written to standard output in the following format:

"%d\n", <number of bytes read>

The current line number shall be set to the address of the last line read in. If *file* is replaced by '!', the rest of the line shall be taken to be a shell command line whose output is to be read. Such a shell command line shall not be remembered as the current pathname.

Substitute Command

Synopsis: (.,.)*s*/RE/replacement/flags

The *s* command shall search each addressed line for an occurrence of the specified RE and replace either the first or all (non-overlapped) matched strings with the *replacement*; see the following description of the *g* suffix. It is an error if the substitution fails on every addressed line. Any character other than <space> or <newline> can be used instead of a <slash> to delimit the RE and the replacement. Within the RE, the RE delimiter itself can be used as a literal character if it is preceded by a <backslash>. The current line shall be set to the address of the last line on which a substitution occurred.

An <ampersand> ('&') appearing in the *replacement* shall be replaced by the string matching the RE on the current line. The special meaning of '&' in this context can be suppressed by preceding it by <backslash>. As a more general feature, the characters '\n', where *n* is a digit, shall be replaced by the text matched by the corresponding back-reference expression. If the corresponding back-reference expression does not match, then the characters '\n' shall be replaced by the empty string. When the character '%' is the only character in the *replacement*, the *replacement* used in the most recent substitute command shall be used as the *replacement* in the current substitute command; if there was no previous substitute command, the use of '%' in this manner shall be an error. The '%' shall lose its special meaning when it is in a replacement string of more than one character or is preceded by a <backslash>. For each <backslash> encountered in scanning *replacement* from beginning to end, the following character shall lose its special meaning (if any). It is unspecified what special meaning is given to any character other than <backslash>, '&', '%', or digits.

A line can be split by substituting a <newline> into it. The application shall ensure it escapes the <newline> in the *replacement* by preceding it by <backslash>. Such substitution cannot be done as part of a *g* or *v* command list. The current line number shall be set to the address of the last line on which a substitution is performed. If no substitution is performed, the current line number shall be unchanged. If a line is split, a substitution shall be considered to have been performed on each of the new lines for the purpose of determining the new current line number. A substitution shall be considered to have been performed even if the replacement string is identical to the string that it replaces.

The application shall ensure that the value of *flags* is zero or more of:

count Substitute for the *count*th occurrence only of the RE found on each addressed line.

- 84815 **g** Globally substitute for all non-overlapping instances of the RE rather than just the first
84816 one. If both **g** and *count* are specified, the results are unspecified.
- 84817 **l** Write to standard output the final line in which a substitution was made. The line shall
84818 be written in the format specified for the **l** command.
- 84819 **n** Write to standard output the final line in which a substitution was made. The line shall
84820 be written in the format specified for the **n** command.
- 84821 **p** Write to standard output the final line in which a substitution was made. The line shall
84822 be written in the format specified for the **p** command.

Copy Command

84823
84824 *Synopsis:* (.,.)*taddress*

84825 The **t** command shall be equivalent to the **m** command, except that a copy of the addressed lines
84826 shall be placed after address *address* (which can be 0); the current line number shall be set to the
84827 address of the last line added.

Undo Command

84828
84829 *Synopsis:* **u**

84830 The **u** command shall nullify the effect of the most recent command that modified anything in
84831 the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **u**, **v**, **G**, or **V** command. All changes
84832 made to the buffer by a **g**, **G**, **v**, or **V** global command shall be undone as a single change; if no
84833 changes were made by the global command (such as with **g/RE/p**), the **u** command shall have
84834 no effect. The current line number shall be set to the value it had immediately before the
84835 command being undone started.

Global Non-Matched Command

84836
84837 *Synopsis:* (1,\$)**v**/*RE/command list*

84838 This command shall be equivalent to the global command **g** except that the lines that are marked
84839 during the first step shall be those for which the line excluding the terminating <newline> does
84840 not match the RE.

Interactive Global Not-Matched Command

84841
84842 *Synopsis:* (1,\$)**V**/*RE/*

84843 This command shall be equivalent to the interactive global command **G** except that the lines that
84844 are marked during the first step shall be those for which the line excluding the terminating
84845 <newline> does not match the RE.

Write Command

84846
84847 *Synopsis:* (1,\$)**w** [*file*]

84848 The **w** command shall write the addressed lines into the file named by the pathname *file*. The
84849 command shall create the file, if it does not exist, or shall replace the contents of the existing file.
84850 The currently remembered pathname shall not be changed unless there is no remembered
84851 pathname. If no pathname is given, the currently remembered pathname, if any, shall be used
84852 (see the **e** and **f** commands); the current line number shall be unchanged. If the command is
84853 successful, the number of bytes written shall be written to standard output, unless the **-s** option
84854 was specified, in the following format:

84855 "%d\n", <number of bytes written>

84856 If *file* begins with '!', the rest of the line shall be taken to be a shell command line whose
 84857 standard input shall be the addressed lines. Such a shell command line shall not be remembered
 84858 as the current pathname. This usage of the write command with '!' shall not be considered as a
 84859 "last **w** command that wrote the entire buffer", as described previously; thus, this alone shall
 84860 not prevent the warning to the user if an attempt is made to destroy the editor buffer via the **e** or
 84861 **q** commands.

84862 Line Number Command

84863 *Synopsis:* (\$) =

84864 The line number of the addressed line shall be written to standard output in the following
 84865 format:

84866 "%d\n", <line number>

84867 The current line number shall be unchanged by this command.

84868 Shell Escape Command

84869 *Synopsis:* !*command*

84870 The remainder of the line after the '!' shall be sent to the command interpreter to be
 84871 interpreted as a shell command line. Within the text of that shell command line, the unescaped
 84872 character '%' shall be replaced with the remembered pathname; if a '!' appears as the first
 84873 character of the command, it shall be replaced with the text of the previous shell command
 84874 executed via '!'. Thus, "!!" shall repeat the previous !*command*. If any replacements of '%' or
 84875 '!' are performed, the modified line shall be written to the standard output before *command* is
 84876 executed. The ! command shall write:

84877 "! \n"

84878 to standard output upon completion, unless the **-s** option is specified. The current line number
 84879 shall be unchanged.

84880 Null Command

84881 *Synopsis:* (.+)

84882 An address alone on a line shall cause the addressed line to be written. A <newline> alone shall
 84883 be equivalent to "+1p". The current line number shall be set to the address of the written line.

84884 EXIT STATUS

84885 The following exit values shall be returned:

- 84886 0 Successful completion without any file or command errors.
- 84887 >0 An error occurred.

84888 CONSEQUENCES OF ERRORS

84889 When an error in the input script is encountered, or when an error is detected that is a
 84890 consequence of the data (not) present in the file or due to an external condition such as a read or
 84891 write error:

- 84892 • If the standard input is a terminal device file, all input shall be flushed, and a new
 84893 command read.

- 84894 • If the standard input is a regular file, *ed* shall terminate with a non-zero exit status.

84895 APPLICATION USAGE

84896 Because of the extremely terse nature of the default error messages, the prudent script writer
84897 begins the *ed* input commands with an **H** command, so that if any errors do occur at least some
84898 clue as to the cause is made available.

84899 In earlier versions of this standard, an obsolescent **–** option was described. This is no longer
84900 specified. Applications should use the **–s** option. Using **–** as a *file* operand now produces
84901 unspecified results. This allows implementations to continue to support the former required
84902 behavior.

84903 EXAMPLES

84904 None.

84905 RATIONALE

84906 The initial description of this utility was adapted from the SVID. It contains some features not
84907 found in Version 7 or BSD-derived systems. Some of the differences between the POSIX and
84908 BSD *ed* utilities include, but need not be limited to:

- 84909 • The BSD **–** option does not suppress the '**!**' prompt after a **!** command.
- 84910 • BSD does not support the special meanings of the '**%**' and '**!**' characters within a **!**
84911 command.
- 84912 • BSD does not support the *addresses* '**;**' and '**r**'.
- 84913 • BSD allows the command/suffix pairs **pp**, **ll**, and so on, which are unspecified in this
84914 volume of POSIX.1-2008.
- 84915 • BSD does not support the '**!**' character part of the **e**, **r**, or **w** commands.
- 84916 • A failed **g** command in BSD sets the line number to the last line searched if there are no
84917 matches.
- 84918 • BSD does not default the *command list* to the **p** command.
- 84919 • BSD does not support the **G**, **h**, **H**, **n**, or **V** commands.
- 84920 • On BSD, if there is no inserted text, the insert command changes the current line to the
84921 referenced line **–1**; that is, the line before the specified line.
- 84922 • On BSD, the *join* command with only a single address changes the current line to that
84923 address.
- 84924 • BSD does not support the **P** command; moreover, in BSD it is synonymous with the **p**
84925 command.
- 84926 • BSD does not support the *undo* of the commands **j**, **m**, **r**, **s**, or **t**.
- 84927 • The Version 7 *ed* command **W**, and the BSD *ed* commands **W**, **wq**, and **z** are not present in
84928 this volume of POSIX.1-2008.

84929 The **–s** option was added to allow the functionality of the removed **–** option in a manner
84930 compatible with the Utility Syntax Guidelines.

84931 In early proposals there was a limit, {ED_FILE_MAX}, that described the historical limitations of
84932 some *ed* utilities in their handling of large files; some of these have had problems with files
84933 larger than 100 000 bytes. It was this limitation that prompted much of the desire to include a
84934 *split* command in this volume of POSIX.1-2008. Since this limit was removed, this volume of
84935 POSIX.1-2008 requires that implementations document the file size limits imposed by *ed* in the

conformance document. The limit {ED_LINE_MAX} was also removed; therefore, the global limit {LINE_MAX} is used for input and output lines.

The manner in which the **l** command writes non-printable characters was changed to avoid the historical backspace-overstrike method. On video display terminals, the overstrike is ambiguous because most terminals simply replace overstruck characters, making the **l** format not useful for its intended purpose of unambiguously understanding the content of the line. The historical <backslash>-escapes were also ambiguous. (The string "a\0011" could represent a line containing those six characters or a line containing the three characters 'a', a byte with a binary value of 1, and a 1.) In the format required here, a <backslash> appearing in the line is written as "\\\" so that the output is truly unambiguous. The method of marking the ends of lines was adopted from the *ex* editor and is required for any line ending in <space> characters; the '\$' is placed on all lines so that a real '\$' at the end of a line cannot be misinterpreted.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this version.

The description of how a NUL is written was removed. The NUL character cannot be in text files, and this volume of POSIX.1-2008 should not dictate behavior in the case of undefined, erroneous input.

Unlike some of the other editing utilities, the filenames accepted by the **E**, **e**, **R**, and **r** commands are not patterns.

Early proposals stated that the **-p** option worked only when standard input was associated with a terminal device. This has been changed to conform to historical implementations, thereby allowing applications to interpose themselves between a user and the *ed* utility.

The form of the substitute command that uses the **n** suffix was limited in some historical documentation (where this was described incorrectly as "backreferencing"). This limit has been omitted because there is no reason why an editor processing lines of {LINE_MAX} length should have this restriction. The command **s/x/2047** should be able to substitute the 2047th occurrence of 'x' on a line.

The use of printing commands with printing suffixes (such as **pn**, **lp**, and so on) was made unspecified because BSD-based systems allow this, whereas System V does not.

Some BSD-based systems exit immediately upon receipt of end-of-file if all of the lines in the file have been deleted. Since this volume of POSIX.1-2008 refers to the **q** command in this instance, such behavior is not allowed.

Some historical implementations returned exit status zero even if command errors had occurred; this is not allowed by this volume of POSIX.1-2008.

Some historical implementations contained a bug that allowed a single <period> to be entered in input mode as <backslash> <period> <newline>. This is not allowed by *ed* because there is no description of escaping any of the characters in input mode; <backslash> characters are entered into the buffer exactly as typed. The typical method of entering a single <period> has been to precede it with another character and then use the substitute command to delete that character.

It is difficult under some modes of some versions of historical operating system terminal drivers to distinguish between an end-of-file condition and terminal disconnect. POSIX.1-2008 does not require implementations to distinguish between the two situations, which permits historical implementations of the *ed* utility on historical platforms to conform. Implementations are encouraged to distinguish between the two, if possible, and take appropriate action on terminal disconnect.

Historically, *ed* accepted a zero address for the **a** and **r** commands in order to insert text at the

start of the edit buffer. When the buffer was empty the command `.=` returned zero. POSIX.1-2008 requires conformance to historical practice.

For consistency with the `a` and `r` commands and better user functionality, the `i` and `c` commands must also accept an address of 0, in which case `0i` is treated as `1i` and likewise for the `c` command.

All of the following are valid addresses:

<code>+++</code>	Three lines after the current line.
<code>/pattern/-</code>	One line before the next occurrence of pattern.
<code>-2</code>	Two lines before the current line.
<code>3 ---- 2</code>	Line one (note the intermediate negative address).
<code>1 2 3</code>	Line six.

Any number of addresses can be provided to commands taking addresses; for example, `"1, 2, 3, 4, 5p"` prints lines 4 and 5, because two is the greatest valid number of addresses accepted by the `print` command. This, in combination with the `<semicolon>` delimiter, permits users to create commands based on ordered patterns in the file. For example, the command `"3; /foo/; +2p"` will display the first line after line 3 that contains the pattern `foo`, plus the next two lines. Note that the address `"3;"` must still be evaluated before being discarded, because the search origin for the `" /foo/"` command depends on this.

Historically, `ed` disallowed address chains, as discussed above, consisting solely of `<comma>` or `<semicolon>` separators; for example, `" , , , "` or `" ; ; ; "` were considered an error. For consistency of address specification, this restriction is removed. The following table lists some of the address forms now possible:

Address	Addr1	Addr2	Status	Comment
<code>7,</code>	7	7	Historical	Valid, but erroneous.
<code>7, 5,</code>	5	5	Historical	
<code>7, 5, 9</code>	5	9	Historical	
<code>7, 9</code>	7	9	Historical	
<code>7, +</code>	7	8	Historical	
<code>,</code>	1	\$	Historical	
<code>, /</code>	1	7	Extension	
<code>,,</code>	\$	\$	Extension	
<code>,;</code>	\$	\$	Extension	
<code>7;</code>	7	7	Historical	
<code>7; 5;</code>	5	5	Historical	
<code>7; 5; 9</code>	5	9	Historical	
<code>7; 5, 9</code>	5	9	Historical	
<code>7; \$; 4</code>	\$	4	Historical	
<code>7; 9</code>	7	9	Historical	
<code>7; +</code>	7	8	Historical	
<code>;</code>	.	\$	Historical	
<code>; 7</code>	.	7	Extension	
<code>;;</code>	\$	\$	Extension	
<code>;;,</code>	\$	\$	Extension	

Historically, `ed` accepted the `'^'` character as an address, in which case it was identical to the `<hyphen>` character. POSIX.1-2008 does not require or prohibit this behavior.

85027 **FUTURE DIRECTIONS**

85028 None.

85029 **SEE ALSO**85030 [Section 1.4](#) (on page 2288), *ex*, *sed*, *sh*, *vi*85031 XBD [Table 5-1](#) (on page 121), [Chapter 8](#) (on page 173), [Section 9.3](#) (on page 183), [Chapter 11](#) (on page 199), [Section 12.2](#) (on page 215)85033 **CHANGE HISTORY**

85034 First released in Issue 2.

85035 **Issue 5**85036 In the OPTIONS section, the meaning of `-s` and `-` is clarified.

85037 A second FUTURE DIRECTION is added.

85038 **Issue 6**

85039 The obsolescent single-minus form is removed.

85040 A second APPLICATION USAGE note is added.

85041 The Open Group Corrigendum U025/2 is applied, correcting the description of the Edit section.

85042 The *ed* utility is updated to align with the IEEE P1003.2b draft standard. This includes addition of the treatment of the SIGQUIT signal, changes to *ed* addressing, and changes to processing when end-of-file is detected and when terminal disconnect is detected.

85045 The normative text is reworded to avoid use of the term “must” for application requirements.

85046 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/22 is applied, adding the text: “Any line modified by the *command list* shall be unmarked.” to the **G** command. This change corresponds to a similar change made to the **g** command in the first version of this standard.

85049 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/7 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.

85051 **Issue 7**85052 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is `' - '`.

85054 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

85055 SD5-XCU-ERN-94 is applied, updating text in the EXTENDED DESCRIPTION where a terminal disconnect is detected (in Commands in *ed*).

85057 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

85058 SD5-XCU-ERN-135 is applied, removing some RATIONALE text that is no longer applicable.

env*Utilities*85059 **NAME**85060 `env` — set the environment for command invocation85061 **SYNOPSIS**85062 `env [-i] [name=value]... [utility [argument...]]`85063 **DESCRIPTION**85064 The *env* utility shall obtain the current environment, modify it according to its arguments, then
85065 invoke the utility named by the *utility* operand with the modified environment.85066 Optional arguments shall be passed to *utility*.85067 If no *utility* operand is specified, the resulting environment shall be written to the standard
85068 output, with one *name=value* pair per line.85069 If the first argument is `'-'`, the results are unspecified.85070 **OPTIONS**85071 The *env* utility shall conform to XBD [Section 12.2](#) (on page 215), except for the unspecified usage
85072 of `'-'`.

85073 The following options shall be supported:

85074 `-i` Invoke *utility* with exactly the environment specified by the arguments; the
85075 inherited environment shall be ignored completely.85076 **OPERANDS**

85077 The following operands shall be supported:

85078 *name=value* Arguments of the form *name=value* shall modify the execution environment, and
85079 shall be placed into the inherited environment before the *utility* is invoked.85080 *utility* The name of the utility to be invoked. If the *utility* operand names any of the
85081 special built-in utilities in [Section 2.14](#) (on page 2334), the results are undefined.85082 *argument* A string to pass as an argument for the invoked utility.85083 **STDIN**

85084 Not used.

85085 **INPUT FILES**

85086 None.

85087 **ENVIRONMENT VARIABLES**85088 The following environment variables shall affect the execution of *env*:85089 *LANG* Provide a default value for the internationalization variables that are unset or null.
85090 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
85091 variables used to determine the values of locale categories.)85092 *LC_ALL* If set to a non-empty string value, override the values of all the other
85093 internationalization variables.85094 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
85095 characters (for example, single-byte as opposed to multi-byte characters in
85096 arguments).85097 *LC_MESSAGES*85098 Determine the locale that should be used to affect the format and contents of
85099 diagnostic messages written to standard error.

85100	XSI	NLSPATH	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
85101		PATH	Determine the location of the <i>utility</i> , as described in XBD Chapter 8 (on page 173).
85102			If <i>PATH</i> is specified as a <i>name=value</i> operand to <i>env</i> , the <i>value</i> given shall be used in
85103			the search for <i>utility</i> .
85104		ASYNCHRONOUS EVENTS	
85105			Default.
85106		STDOUT	
85107			If no <i>utility</i> operand is specified, each <i>name=value</i> pair in the resulting environment shall be
85108			written in the form:
85109			"%s=%s\n", <name>, <value>
85110			If the <i>utility</i> operand is specified, the <i>env</i> utility shall not write to standard output.
85111		STDERR	
85112			The standard error shall be used only for diagnostic messages.
85113		OUTPUT FILES	
85114			None.
85115		EXTENDED DESCRIPTION	
85116			None.
85117		EXIT STATUS	
85118			If <i>utility</i> is invoked, the exit status of <i>env</i> shall be the exit status of <i>utility</i> ; otherwise, the <i>env</i>
85119			utility shall exit with one of the following values:
85120		0	The <i>env</i> utility completed successfully.
85121		1–125	An error occurred in the <i>env</i> utility.
85122		126	The utility specified by <i>utility</i> was found but could not be invoked.
85123		127	The utility specified by <i>utility</i> could not be found.
85124		CONSEQUENCES OF ERRORS	
85125			Default.
85126		APPLICATION USAGE	
85127			The <i>command</i> , <i>env</i> , <i>nice</i> , <i>nohup</i> , <i>time</i> , and <i>xargs</i> utilities have been specified to use exit code 127 if
85128			an error occurs so that applications can distinguish “failure to find a utility” from “invoked
85129			utility exited with an error indication”. The value 127 was chosen because it is not commonly
85130			used for other meanings; most utilities use small values for “normal error conditions” and the
85131			values above 128 can be confused with termination due to receipt of a signal. The value 126 was
85132			chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
85133			scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
85134			between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
85135			<i>exec</i> the utility fail with [ENOENT], and uses 126 when any attempt to <i>exec</i> the utility fails for
85136			any other reason.
85137			Historical implementations of the <i>env</i> utility use the <i>execvp()</i> or <i>execlp()</i> functions defined in the
85138			System Interfaces volume of POSIX.1-2008 to invoke the specified utility; this provides better
85139			performance and keeps users from having to escape characters with special meaning to the shell.
85140			Therefore, shell functions, special built-ins, and built-ins that are only provided by the shell are
85141			not found.

EXAMPLES

The following command:

```
env -i PATH=/mybin:"$PATH" $(getconf V7_ENV) mygrep xyz myfile
```

invokes the command *mygrep* with a new *PATH* value as the only entry in its environment other than any variables required by the implementation for conformance. In this case, *PATH* is used to locate *mygrep*, which is expected to reside in */mybin*.

RATIONALE

As with all other utilities that invoke other utilities, this volume of POSIX.1-2008 only specifies what *env* does with standard input, standard output, standard error, input files, and output files. If a utility is executed, it is not constrained by the specification of input and output by *env*.

The *-i* option was added to allow the functionality of the removed *-* option in a manner compatible with the Utility Syntax Guidelines. It is possible to create a non-conforming environment using the *-i* option, as it may remove environment variables required by the implementation for conformance. The following will preserve these environment variables as well as preserve the *PATH* for conforming utilities:

```
IFS='
# The preceding value should be <space><tab><newline>.
# Set IFS to its default value.

set -f
# disable pathname expansion

\unalias -a
# Unset all possible aliases.
# Note that unalias is escaped to prevent an alias
# being used for unalias.
# This step is not strictly necessary, since aliases are not inherited,
# and the ENV environment variable is only used by interactive shells,
# the only way any aliases can exist in a script is if it defines them
# itself.

unset -f env getconf
# Ensure env and getconf are not user functions.

env -i $(getconf V7_ENV) PATH="$(getconf PATH)" command
```

Some have suggested that *env* is redundant since the same effect is achieved by:

```
name=value ... utility [ argument ... ]
```

The example is equivalent to *env* when an environment variable is being added to the environment of the command, but not when the environment is being set to the given value. The *env* utility also writes out the current environment if invoked without arguments. There is sufficient functionality beyond what the example provides to justify inclusion of *env*.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.14](#) (on page 2334), [Section 2.5](#) (on page 2301)

XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

85185 **CHANGE HISTORY**

85186 First released in Issue 2.

85187 **Issue 7**85188 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first
85189 argument is ' - ' .85190 Austin Group Interpretation 1003.1-2001 #047 is applied, providing RATIONALE on how to use
85191 the *env* utility to preserve a conforming environment.

85192 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

85193 The EXAMPLES section is revised to change the use of *env* -i.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

85194 NAME

85195 ex — text editor

85196 SYNOPSIS

85197 UP `ex [-rR] [-s|-v] [-c command] [-t tagstring] [-w size] [file...]`

85198 DESCRIPTION

85199 The *ex* utility is a line-oriented text editor. There are two other modes of the editor—open and
 85200 visual—in which screen-oriented editing is available. This is described more fully by the **ex open**
 85201 and **visual** commands and in *vi*.

85202 If an operand is *'-'*, the results are unspecified.

85203 This section uses the term *edit buffer* to describe the current working text. No specific
 85204 implementation is implied by this term. All editing changes are performed on the edit buffer,
 85205 and no changes to it shall affect any file until an editor command writes the file.

85206 Certain terminals do not have all the capabilities necessary to support the complete *ex* definition,
 85207 such as the full-screen editing commands (*visual mode* or *open mode*). When these commands
 85208 cannot be supported on such terminals, this condition shall not produce an error message such
 85209 as “not an editor command” or report a syntax error. The implementation may either accept the
 85210 commands and produce results on the screen that are the result of an unsuccessful attempt to
 85211 meet the requirements of this volume of POSIX.1-2008 or report an error describing the terminal-
 85212 related deficiency.

85213 OPTIONS

85214 The *ex* utility shall conform to XBD Section 12.2 (on page 215), except for the unspecified usage
 85215 of *'-'*, and that *'+'* may be recognized as an option delimiter as well as *'-'*.

85216 The following options shall be supported:

85217 **-c *command*** Specify an initial command to be executed in the first edit buffer loaded from an
 85218 existing file (see the EXTENDED DESCRIPTION section). Implementations may
 85219 support more than a single **-c** option. In such implementations, the specified
 85220 commands shall be executed in the order specified on the command line.

85221 **-r** Recover the named files (see the EXTENDED DESCRIPTION section). Recovery
 85222 information for a file shall be saved during an editor or system crash (for example,
 85223 when the editor is terminated by a signal which the editor can catch), or after the
 85224 use of an *ex preserve* command.

85225 A *crash* in this context is an unexpected failure of the system or utility that requires
 85226 restarting the failed system or utility. A system crash implies that any utilities
 85227 running at the time also crash. In the case of an editor or system crash, the number
 85228 of changes to the edit buffer (since the most recent **preserve** command) that will be
 85229 recovered is unspecified.

85230 If no *file* operands are given and the **-t** option is not specified, all other options, the
 85231 *EXINIT* variable, and any **.exrc** files shall be ignored; a list of all recoverable files
 85232 available to the invoking user shall be written, and the editor shall exit normally
 85233 without further action.

85234 **-R** Set **readonly** edit option.

85235 **-s** Prepare *ex* for batch use by taking the following actions:

- 85236 • Suppress writing prompts and informational (but not diagnostic) messages.
- 85237 • Ignore the value of *TERM* and any implementation default terminal type and
- 85238 assume the terminal is a type incapable of supporting open or visual modes;
- 85239 see the **visual** command and the description of *vi*.
- 85240 • Suppress the use of the *EXINIT* environment variable and the reading of any
- 85241 **.exrc** file; see the EXTENDED DESCRIPTION section.
- 85242 • Suppress autoindentation, ignoring the value of the **autoindent** edit option.
- 85243 **-t tagstring** Edit the file containing the specified *tagstring*; see *ctags*. The *tags* feature
- 85244 represented by **-t tagstring** and the **tag** command is optional. It shall be provided
- 85245 on any system that also provides a conforming implementation of *ctags*; otherwise,
- 85246 the use of **-t** produces undefined results. On any system, it shall be an error to
- 85247 specify more than a single **-t** option.
- 85248 **-v** Begin in visual mode (see *vi*).
- 85249 **-w size** Set the value of the *window* editor option to *size*.

85250 OPERANDS

85251 The following operand shall be supported:

85252 *file* A pathname of a file to be edited.

85253 STDIN

85254 The standard input consists of a series of commands and input text, as described in the

85255 EXTENDED DESCRIPTION section. The implementation may limit each line of standard input

85256 to a length of {LINE_MAX}.

85257 If the standard input is not a terminal device, it shall be as if the **-s** option had been specified.

85258 If a read from the standard input returns an error, or if the editor detects an end-of-file condition

85259 from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

85260 INPUT FILES

85261 Input files shall be text files or files that would be text files except for an incomplete last line that

85262 is not longer than {LINE_MAX}-1 bytes in length and contains no NUL characters. By default,

85263 any incomplete last line shall be treated as if it had a trailing <newline>. The editing of other

85264 forms of files may optionally be allowed by *ex* implementations.

85265 The **.exrc** files and source files shall be text files consisting of *ex* commands; see the EXTENDED

85266 DESCRIPTION section.

85267 By default, the editor shall read lines from the files to be edited without interpreting any of those

85268 lines as any form of editor command.

85269 ENVIRONMENT VARIABLES

85270 The following environment variables shall affect the execution of *ex*:

- 85271 **COLUMNS** Override the system-selected horizontal screen size. See XBD Chapter 8 (on page
- 85272 173) for valid values and results when it is unset or null.
- 85273 **EXINIT** Determine a list of *ex* commands that are executed on editor start-up. See the
- 85274 EXTENDED DESCRIPTION section for more details of the initialization phase.
- 85275 **HOME** Determine a pathname of a directory that shall be searched for an editor start-up
- 85276 file named **.exrc**; see the EXTENDED DESCRIPTION section.

85277	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
85278		
85279		
85280	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
85281		
85282	<i>LC_COLLATE</i>	Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.
85283		
85284		
85285	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the classification of characters as uppercase or lowercase letters, the case conversion of letters, and the detection of word boundaries.
85286		
85287		
85288		
85289		
85290	<i>LC_MESSAGES</i>	Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
85291		
85292		
85293	<i>LINES</i>	Override the system-selected vertical screen size, used as the number of lines in a screenful and the vertical screen size in visual mode. See XBD Chapter 8 (on page 173) for valid values and results when it is unset or null.
85294		
85295		
85296	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
85297	<i>PATH</i>	Determine the search path for the shell command specified in the <i>ex</i> editor commands ! , shell , read , and write , and the open and visual mode command ! ; see the description of command search and execution in Section 2.9.1.1 (on page 2317).
85298		
85299		
85300	<i>SHELL</i>	Determine the preferred command line interpreter for use as the default value of the shell edit option.
85301		
85302	<i>TERM</i>	Determine the name of the terminal type. If this variable is unset or null, an unspecified default terminal type shall be used.
85303		
85304	ASYNCHRONOUS EVENTS	
85305	The following term is used in this and following sections to specify command and asynchronous event actions:	
85306		
85307	<i>complete write</i>	
85308		A complete write is a write of the entire contents of the edit buffer to a file of a type other than a terminal device, or the saving of the edit buffer caused by the user executing the <i>ex</i> preserve command. Writing the contents of the edit buffer to a temporary file that will be removed when the editor exits shall not be considered a complete write.
85309		
85310		
85311		
85312		
85313	The following actions shall be taken upon receipt of signals:	
85314	<i>SIGINT</i>	If the standard input is not a terminal device, <i>ex</i> shall not write the file or return to command or text input mode, and shall exit with a non-zero exit status.
85315		
85316		Otherwise, if executing an open or visual text input mode command, <i>ex</i> in receipt of <i>SIGINT</i> shall behave identically to its receipt of the <ESC> character.
85317		
85318		Otherwise:

Utilities

ex

- 85319 1. If executing an *ex* text input mode command, all input lines that have been
85320 completely entered shall be resolved into the edit buffer, and any partially
85321 entered line shall be discarded.
- 85322 2. If there is a currently executing command, it shall be aborted and a message
85323 displayed. Unless otherwise specified by the *ex* or *vi* command descriptions,
85324 it is unspecified whether any lines modified by the executing command
85325 appear modified, or as they were before being modified by the executing
85326 command, in the buffer.
- 85327 If the currently executing command was a motion command, its associated
85328 command shall be discarded.
- 85329 3. If in open or visual command mode, the terminal shall be alerted.
- 85330 4. The editor shall then return to command mode.
- 85331 **SIGCONT** The screen shall be refreshed if in open or visual mode.
- 85332 **SIGHUP** If the edit buffer has been modified since the last complete write, *ex* shall attempt
85333 to save the edit buffer so that it can be recovered later using the **-r** option or the *ex*
85334 **recover** command. The editor shall not write the file or return to command or text
85335 input mode, and shall terminate with a non-zero exit status.
- 85336 **SIGTERM** Refer to **SIGHUP**.
- 85337 The action taken for all other signals is unspecified.
- 85338 **STDOUT**
- 85339 The standard output shall be used only for writing prompts to the user, for informational
85340 messages, and for writing lines from the file.
- 85341 **STDERR**
- 85342 The standard error shall be used only for diagnostic messages.
- 85343 **OUTPUT FILES**
- 85344 The output from *ex* shall be text files.
- 85345 **EXTENDED DESCRIPTION**
- 85346 Only the *ex* mode of the editor is described in this section. See *vi* for additional editing
85347 capabilities available in *ex*.
- 85348 When an error occurs, *ex* shall write a message. If the terminal supports a standout mode (such
85349 as inverse video), the message shall be written in standout mode. If the terminal does not
85350 support a standout mode, and the edit option **errorbells** is set, an alert action shall precede the
85351 error message.
- 85352 By default, *ex* shall start in command mode, which shall be indicated by a **:** prompt; see the
85353 **prompt** command. Text input mode can be entered by the **append**, **insert**, or **change** commands;
85354 it can be exited (and command mode re-entered) by typing a **<period>** (**'.'**) alone at the
85355 beginning of a line.

Initialization in ex and vi

The following symbols are used in this and following sections to specify locations in the edit buffer:

alternate and current pathnames

Two pathnames, named *current* and *alternate*, are maintained by the editor. Any *ex* commands that take filenames as arguments shall set them as follows:

1. If a *file* argument is specified to the *ex* **edit**, **ex**, or **recover** commands, or if an *ex* **tag** command replaces the contents of the edit buffer.
 - a. If the command replaces the contents of the edit buffer, the current pathname shall be set to the *file* argument or the file indicated by the tag, and the alternate pathname shall be set to the previous value of the current pathname.
 - b. Otherwise, the alternate pathname shall be set to the *file* argument.
2. If a *file* argument is specified to the *ex* **next** command:
 - a. If the command replaces the contents of the edit buffer, the current pathname shall be set to the first *file* argument, and the alternate pathname shall be set to the previous value of the current pathname.
3. If a *file* argument is specified to the *ex* **file** command, the current pathname shall be set to the *file* argument, and the alternate pathname shall be set to the previous value of the current pathname.
4. If a *file* argument is specified to the *ex* **read** and **write** commands (that is, when reading or writing a file, and not to the program named by the **shell** edit option), or a *file* argument is specified to the *ex* **xit** command:
 - a. If the current pathname has no value, the current pathname shall be set to the *file* argument.
 - b. Otherwise, the alternate pathname shall be set to the *file* argument.

If the alternate pathname is set to the previous value of the current pathname when the current pathname had no previous value, then the alternate pathname shall have no value as a result.

current line

The line of the edit buffer referenced by the cursor. Each command description specifies the current line after the command has been executed, as the *current line value*. When the edit buffer contains no lines, the current line shall be zero; see [Addressing in ex](#) (on page 2644).

current column

The current display line column occupied by the cursor. (The columns shall be numbered beginning at 1.) Each command description specifies the current column after the command has been executed, as the *current column value*. This column is an *ideal* column that is remembered over the lifetime of the editor. The actual display line column upon which the cursor rests may be different from the current column; see the cursor positioning discussion in [Command Descriptions in vi](#) (on page 3310).

set to non-<blank>

A description for a current column value, meaning that the current column shall be set to the last display line column on which is displayed any part of the first non-<blank> of the line. If the line has no non-<blank> non-<newline> characters, the current column shall be set to the last display line column on which is displayed any part of the last non-<newline>

85400 character in the line. If the line is empty, the current column shall be set to column position
85401 1.

85402 The length of lines in the edit buffer may be limited to {LINE_MAX} bytes. In open and visual
85403 mode, the length of lines in the edit buffer may be limited to the number of characters that will
85404 fit in the display. If either limit is exceeded during editing, an error message shall be written. If
85405 either limit is exceeded by a line read in from a file, an error message shall be written and the
85406 edit session may be terminated.

85407 If the editor stops running due to any reason other than a user command, and the edit buffer has
85408 been modified since the last complete write, it shall be equivalent to a SIGHUP asynchronous
85409 event. If the system crashes, it shall be equivalent to a SIGHUP asynchronous event.

85410 During initialization (before the first file is copied into the edit buffer or any user commands
85411 from the terminal are processed) the following shall occur:

- 85412 1. If the environment variable *EXINIT* is set, the editor shall execute the *ex* commands
85413 contained in that variable.
- 85414 2. If the *EXINIT* variable is not set, and all of the following are true:
85415 a. The *HOME* environment variable is not null and not empty.
85416 b. The file *.exrc* in the directory referred to by the *HOME* environment variable:
85417 i. Exists
85418 ii. Is owned by the same user ID as the real user ID of the process or the
85419 process has appropriate privileges
85420 iii. Is not writable by anyone other than the owner

85421 the editor shall execute the *ex* commands contained in that file.

- 85422 3. If and only if all of the following are true:
85423 a. The current directory is not referred to by the *HOME* environment variable.
85424 b. A command in the *EXINIT* environment variable or a command in the *.exrc* file in
85425 the directory referred to by the *HOME* environment variable sets the editor option
85426 *exrc*.
85427 c. The *.exrc* file in the current directory:
85428 i. Exists
85429 ii. Is owned by the same user ID as the real user ID of the process, or by one of
85430 a set of implementation-defined user IDs
85431 iii. Is not writable by anyone other than the owner

85432 the editor shall attempt to execute the *ex* commands contained in that file.

85433 Lines in any *.exrc* file that are blank lines shall be ignored. If any *.exrc* file exists, but is not read
85434 for ownership or permission reasons, it shall be an error.

85435 After the *EXINIT* variable and any *.exrc* files are processed, the first file specified by the user
85436 shall be edited, as follows:

- 85437 1. If the user specified the *-t* option, the effect shall be as if the *ex tag* command was entered
85438 with the specified argument, with the exception that if tag processing does not result in a
85439 file to edit, the effect shall be as described in step 3. below.

2. Otherwise, if the user specified any command line *file* arguments, the effect shall be as if the *ex* **edit** command was entered with the first of those arguments as its *file* argument.
3. Otherwise, the effect shall be as if the *ex* **edit** command was entered with a nonexistent filename as its *file* argument. It is unspecified whether this action shall set the current pathname. In an implementation where this action does not set the current pathname, any editor command using the current pathname shall fail until an editor command sets the current pathname.

If the **-r** option was specified, the first time a file in the initial argument list or a file specified by the **-t** option is edited, if recovery information has previously been saved about it, that information shall be recovered and the editor shall behave as if the contents of the edit buffer have already been modified. If there are multiple instances of the file to be recovered, the one most recently saved shall be recovered, and an informational message that there are previous versions of the file that can be recovered shall be written. If no recovery information about a file is available, an informational message to this effect shall be written, and the edit shall proceed as usual.

If the **-c** option was specified, the first time a file that already exists (including a file that might not exist but for which recovery information is available, when the **-r** option is specified) replaces or initializes the contents of the edit buffer, the current line shall be set to the last line of the edit buffer, the current column shall be set to non-**<blank>**, and the *ex* commands specified with the **-c** option shall be executed. In this case, the current line and current column shall not be set as described for the command associated with the replacement or initialization of the edit buffer contents. However, if the **-t** option or a **tag** command is associated with this action, the **-c** option commands shall be executed and then the movement to the tag shall be performed.

The current argument list shall initially be set to the filenames specified by the user on the command line. If no filenames are specified by the user, the current argument list shall be empty. If the **-t** option was specified, it is unspecified whether any filename resulting from tag processing shall be prepended to the current argument list. In the case where the filename is added as a prefix to the current argument list, the current argument list reference shall be set to that filename. In the case where the filename is not added as a prefix to the current argument list, the current argument list reference shall logically be located before the first of the filenames specified on the command line (for example, a subsequent *ex* **next** command shall edit the first filename from the command line). If the **-t** option was not specified, the current argument list reference shall be to the first of the filenames on the command line.

Addressing in *ex*

Addressing in *ex* relates to the current line and the current column; the address of a line is its 1-based line number, the address of a column is its 1-based count from the beginning of the line. Generally, the current line is the last line affected by a command. The current line number is the address of the current line. In each command description, the effect of the command on the current line number and the current column is described.

Addresses are constructed as follows:

1. The character ' .' (period) shall address the current line.
2. The character ' \$ ' shall address the last line of the edit buffer.
3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
4. The address " ' x " refers to the line marked with the mark name character ' x ', which shall be a lowercase letter from the portable character set, the backquote character, or the single-quote character. It shall be an error if the line that was marked is not currently

present in the edit buffer or the mark has not been set. Lines can be marked with the *ex mark* or *k* commands, or the *vi m* command.

5. A regular expression enclosed by <slash> characters ('/') shall address the first line found by searching forwards from the line following the current line toward the end of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the regular expression. As stated in [Regular Expressions in ex](#) (on page 2675), an address consisting of a null regular expression delimited by <slash> characters ("/") shall address the next line for which the line excluding the terminating <newline> matches the last regular expression encountered. In addition, the second <slash> can be omitted at the end of a command line. If the **wrapscan** edit option is set, the search shall wrap around to the beginning of the edit buffer and continue up to and including the current line, so that the entire edit buffer is searched. Within the regular expression, the sequence "\/" shall represent a literal <slash> instead of the regular expression delimiter.
6. A regular expression enclosed in <question-mark> characters ('?') shall address the first line found by searching backwards from the line preceding the current line toward the beginning of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the regular expression. An address consisting of a null regular expression delimited by <question-mark> characters ("??") shall address the previous line for which the line excluding the terminating <newline> matches the last regular expression encountered. In addition, the second <question-mark> can be omitted at the end of a command line. If the **wrapscan** edit option is set, the search shall wrap around from the beginning of the edit buffer to the end of the edit buffer and continue up to and including the current line, so that the entire edit buffer is searched. Within the regular expression, the sequence "\?" shall represent a literal <question-mark> instead of the RE delimiter.
7. A <plus-sign> ('+') or a <minus-sign> ('-') followed by a decimal number shall address the current line plus or minus the number. A '+' or '-' not followed by a decimal number shall address the current line plus or minus 1.

Addresses can be followed by zero or more address offsets, optionally <blank>-separated. Address offsets are constructed as follows:

1. A '+' or '-' immediately followed by a decimal number shall add (subtract) the indicated number of lines to (from) the address. A '+' or '-' not followed by a decimal number shall add (subtract) 1 to (from) the address.
2. A decimal number shall add the indicated number of lines to the address.

It shall not be an error for an intermediate address value to be less than zero or greater than the last line in the edit buffer. It shall be an error for the final address value to be less than zero or greater than the last line in the edit buffer.

Commands take zero, one, or two addresses; see the descriptions of *1addr* and *2addr* in [Command Descriptions in ex](#) (on page 2651). If more than the required number of addresses are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than the required number of addresses are provided to a command, the addresses specified first shall be evaluated and then discarded until the maximum number of valid addresses remain.

Addresses shall be separated from each other by a <comma> (',') or a <semicolon> (';'). If no address is specified before or after a <comma> or <semicolon> separator, it shall be as if the address of the current line was specified before or after the separator. In the case of a <semicolon> separator, the current line ('.') shall be set to the first address, and only then will

85533 the next address be calculated. This feature can be used to determine the starting line for
85534 forwards and backwards searches (see rules 5. and 6.).

85535 A <percent-sign> (' % ') shall be equivalent to entering the two addresses "1, \$".

85536 Any delimiting <blank> characters between addresses, address separators, or address offsets
85537 shall be discarded.

85538 Command Line Parsing in ex

85539 The following symbol is used in this and following sections to describe parsing behavior:

85540 *escape* If a character is referred to as "<backslash>-escaped" or "<control>-V-escaped", it
85541 shall mean that the character acquired or lost a special meaning by virtue of being
85542 preceded, respectively, by a <backslash> or <control>-V character. Unless
85543 otherwise specified, the escaping character shall be discarded at that time and shall
85544 not be further considered for any purpose.

85545 Command-line parsing shall be done in the following steps. For each step, characters already
85546 evaluated shall be ignored; that is, the phrase "leading character" refers to the next character
85547 that has not yet been evaluated.

- 85548 1. Leading <colon> characters shall be skipped.
- 85549 2. Leading <blank> characters shall be skipped.
- 85550 3. If the leading character is a double-quote character, the characters up to and including the
85551 next non-<backslash>-escaped <newline> shall be discarded, and any subsequent
85552 characters shall be parsed as a separate command.
- 85553 4. Leading characters that can be interpreted as addresses shall be evaluated; see
85554 [Addressing in ex](#) (on page 2644).
- 85555 5. Leading <blank> characters shall be skipped.
- 85556 6. If the next character is a <vertical-line> character or a <newline>:
85557 a. If the next character is a <newline>:
85558 i. If *ex* is in open or visual mode, the current line shall be set to the last
85559 address specified, if any.
85560 ii. Otherwise, if the last command was terminated by a <vertical-line>
85561 character, no action shall be taken; for example, the command
85562 "||<newline>" shall execute two implied commands, not three.
85563 iii. Otherwise, step 6.b. shall apply.
- 85564 b. Otherwise, the implied command shall be the **print** command. The last #, **p**, and **l**
85565 flags specified to any *ex* command shall be remembered and shall apply to this
85566 implied command. Executing the *ex* **number**, **print**, or **list** command shall set the
85567 remembered flags to #, nothing, and **l**, respectively, plus any other flags specified
85568 for that execution of the **number**, **print**, or **list** command.

85569 If *ex* is not currently performing a **global** or **v** command, and no address or count
85570 is specified, the current line shall be incremented by 1 before the command is
85571 executed. If incrementing the current line would result in an address past the last
85572 line in the edit buffer, the command shall fail, and the increment shall not happen.

- c. The <newline> or <vertical-line> character shall be discarded and any subsequent characters shall be parsed as a separate command.
7. The command name shall be comprised of the next character (if the character is not alphabetic), or the next character and any subsequent alphabetic characters (if the character is alphabetic), with the following exceptions:
- a. Commands that consist of any prefix of the characters in the command name **delete**, followed immediately by any of the characters 'l', 'p', '+', '-', or '#', shall be interpreted as a **delete** command, followed by a <blank>, followed by the characters that were not part of the prefix of the **delete** command. The maximum number of characters shall be matched to the command name **delete**; for example, "del" shall not be treated as "de" followed by the flag l.
 - b. Commands that consist of the character 'k', followed by a character that can be used as the name of a mark, shall be equivalent to the mark command followed by a <blank>, followed by the character that followed the 'k'.
 - c. Commands that consist of the character 's', followed by characters that could be interpreted as valid options to the **s** command, shall be the equivalent of the **s** command, without any pattern or replacement values, followed by a <blank>, followed by the characters after the 's'.
8. The command name shall be matched against the possible command names, and a command name that contains a prefix matching the characters specified by the user shall be the executed command. In the case of commands where the characters specified by the user could be ambiguous, the executed command shall be as follows:

a	append	n	next	t	t
c	change	p	print	u	undo
ch	change	pr	print	un	undo
e	edit	r	read	v	v
m	move	re	read	w	write
ma	mark	s	s		

- Implementation extensions with names causing similar ambiguities shall not be checked for a match until all possible matches for commands specified by POSIX.1-2008 have been checked.
9. If the command is a **!** command, or if the command is a **read** command followed by zero or more <blank> characters and a **!**, or if the command is a **write** command followed by one or more <blank> characters and a **!**, the rest of the command shall include all characters up to a non-<backslash>-escaped <newline>. The <newline> shall be discarded and any subsequent characters shall be parsed as a separate *ex* command.
10. Otherwise, if the command is an **edit**, **ex**, or **next** command, or a **visual** command while in open or visual mode, the next part of the command shall be parsed as follows:
- a. Any '!' character immediately following the command shall be skipped and be part of the command.
 - b. Any leading <blank> characters shall be skipped and be part of the command.
 - c. If the next character is a '+', characters up to the first non-<backslash>-escaped <newline> or non-<backslash>-escaped <blank> shall be skipped and be part of the command.

- 85617 d. The rest of the command shall be determined by the steps specified in paragraph
85618 12.
- 85619 11. Otherwise, if the command is a **global**, **open**, **s**, or **v** command, the next part of the
85620 command shall be parsed as follows:
- 85621 a. Any leading <blank> characters shall be skipped and be part of the command.
- 85622 b. If the next character is not an alphanumeric, double-quote, <newline>,
85623 <backslash>, or <vertical-line> character:
- 85624 i. The next character shall be used as a command delimiter.
- 85625 ii. If the command is a **global**, **open**, or **v** command, characters up to the first
85626 non-<backslash>-escaped <newline>, or first non-<backslash>-escaped
85627 delimiter character, shall be skipped and be part of the command.
- 85628 iii. If the command is an **s** command, characters up to the first
85629 non-<backslash>-escaped <newline>, or second non-<backslash>-escaped
85630 delimiter character, shall be skipped and be part of the command.
- 85631 c. If the command is a **global** or **v** command, characters up to the first
85632 non-<backslash>-escaped <newline> shall be skipped and be part of the
85633 command.
- 85634 d. Otherwise, the rest of the command shall be determined by the steps specified in
85635 paragraph 12.
- 85636 12. Otherwise:
- 85637 a. If the command was a **map**, **unmap**, **abbreviate**, or **unabbreviate** command,
85638 characters up to the first non-<control>-V-escaped <newline>, <vertical-line>, or
85639 double-quote character shall be skipped and be part of the command.
- 85640 b. Otherwise, characters up to the first non-<backslash>-escaped <newline>,
85641 <vertical-line>, or double-quote character shall be skipped and be part of the
85642 command.
- 85643 c. If the command was an **append**, **change**, or **insert** command, and the step 12.b.
85644 ended at a <vertical-line> character, any subsequent characters, up to the next
85645 non-<backslash>-escaped <newline> shall be used as input text to the command.
- 85646 d. If the command was ended by a double-quote character, all subsequent characters,
85647 up to the next non-<backslash>-escaped <newline>, shall be discarded.
- 85648 e. The terminating <newline> or <vertical-line> character shall be discarded and any
85649 subsequent characters shall be parsed as a separate *ex* command.
- 85650 Command arguments shall be parsed as described by the Synopsis and Description of each
85651 individual *ex* command. This parsing shall not be <blank>-sensitive, except for the **!** argument,
85652 which must follow the command name without intervening <blank> characters, and where it
85653 would otherwise be ambiguous. For example, *count* and *flag* arguments need not be
85654 <blank>-separated because "*d22p*" is not ambiguous, but *file* arguments to the *ex* **next**
85655 command must be separated by one or more <blank> characters. Any <blank> in command
85656 arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands can be
85657 <control>-V-escaped, in which case the <blank> shall not be used as an argument delimiter. Any
85658 <blank> in the command argument for any other command can be <backslash>-escaped, in
85659 which case that <blank> shall not be used as an argument delimiter.
- 85660 Within command arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands,

any character can be <control>-V-escaped. All such escaped characters shall be treated literally and shall have no special meaning. Within command arguments for all other *ex* commands that are not regular expressions or replacement strings, any character that would otherwise have a special meaning can be <backslash>-escaped. Escaped characters shall be treated literally, without special meaning as shell expansion characters or '!', '%', and '#' expansion characters. See [Regular Expressions in ex](#) (on page 2675) and [Replacement Strings in ex](#) (on page 2676) for descriptions of command arguments that are regular expressions or replacement strings.

Non-<backslash>-escaped '%' characters appearing in *file* arguments to any *ex* command shall be replaced by the current pathname; unescaped '#' characters shall be replaced by the alternate pathname. It shall be an error if '%' or '#' characters appear unescaped in an argument and their corresponding values are not set.

Non-<backslash>-escaped '!' characters in the arguments to either the *ex* ! command or the open and visual mode ! command, or in the arguments to the *ex* read command, where the first non-<blank> after the command name is a '!' character, or in the arguments to the *ex* write command where the command name is followed by one or more <blank> characters and the first non-<blank> after the command name is a '!' character, shall be replaced with the arguments to the last of those three commands as they appeared after all unescaped '%', '#', and '!' characters were replaced. It shall be an error if '!' characters appear unescaped in one of these commands and there has been no previous execution of one of these commands.

If an error occurs during the parsing or execution of an *ex* command:

- An informational message to this effect shall be written. Execution of the *ex* command shall stop, and the cursor (for example, the current line and column) shall not be further modified.
- If the *ex* command resulted from a map expansion, all characters from that map expansion shall be discarded, except as otherwise specified by the **map** command.
- Otherwise, if the *ex* command resulted from the processing of an *EXINIT* environment variable, a **.exrc** file, a **:source** command, a **-c** option, or a **+command** specified to an *ex* **edit**, **ex**, **next**, or **visual** command, no further commands from the source of the commands shall be executed.
- Otherwise, if the *ex* command resulted from the execution of a buffer or a **global** or **v** command, no further commands caused by the execution of the buffer or the **global** or **v** command shall be executed.
- Otherwise, if the *ex* command was not terminated by a <newline>, all characters up to and including the next non-<backslash>-escaped <newline> shall be discarded.

Input Editing in ex

The following symbol is used in this and the following sections to specify command actions:

word In the POSIX locale, a word consists of a maximal sequence of letters, digits, and underscores, delimited at both ends by characters other than letters, digits, or underscores, or by the beginning or end of a line or the edit buffer.

When accepting input characters from the user, in either *ex* command mode or *ex* text input mode, *ex* shall enable canonical mode input processing, as defined in the System Interfaces volume of POSIX.1-2008.

85704 If in *ex* text input mode:

- 85705 1. If the **number** edit option is set, *ex* shall prompt for input using the line number that
85706 would be assigned to the line if it is entered, in the format specified for the *ex* **number**
85707 command.
- 85708 2. If the **autoindent** edit option is set, *ex* shall prompt for input using **autoindent** characters,
85709 as described by the **autoindent** edit option. **autoindent** characters shall follow the line
85710 number, if any.

85711 If in *ex* command mode:

- 85712 1. If the **prompt** edit option is set, input shall be prompted for using a single **?** character;
85713 otherwise, there shall be no prompt.

85714 The input characters in the following sections shall have the following effects on the input line.

85715 Scroll

85716 *Synopsis:* `eof`

85717 See the description of the *stty* `eof` character in *stty*.

85718 If in *ex* command mode:

85719 If the `eof` character is the first character entered on the line, the line shall be evaluated as if
85720 it contained two characters: a <control>-D and a <newline>.

85721 Otherwise, the `eof` character shall have no special meaning.

85722 If in *ex* text input mode:

85723 If the cursor follows an **autoindent** character, the **autoindent** characters in the line shall be
85724 modified so that a part of the next text input character will be displayed on the first
85725 column in the line after the previous **shiftwidth** edit option column boundary, and the
85726 user shall be prompted again for input for the same line.

85727 Otherwise, if the cursor follows a '0', which follows an **autoindent** character, and the '0'
85728 was the previous text input character, the '0' and all **autoindent** characters in the line
85729 shall be discarded, and the user shall be prompted again for input for the same line.

85730 Otherwise, if the cursor follows a '^', which follows an **autoindent** character, and the '^'
85731 was the previous text input character, the '^' and all **autoindent** characters in the line
85732 shall be discarded, and the user shall be prompted again for input for the same line. In
85733 addition, the **autoindent** level for the next input line shall be derived from the same line
85734 from which the **autoindent** level for the current input line was derived.

85735 Otherwise, if there are no **autoindent** or text input characters in the line, the `eof` character
85736 shall be discarded.

85737 Otherwise, the `eof` character shall have no special meaning.

<newline>

Synopsis: <newline>
 <control>-J

If in *ex* command mode:

Cause the command line to be parsed; <control>-J shall be mapped to the <newline> for this purpose.

If in *ex* text input mode:

Terminate the current line. If there are no characters other than **autoindent** characters on the line, all characters on the line shall be discarded.

Prompt for text input on a new line after the current line. If the **autoindent** edit option is set, an appropriate number of **autoindent** characters shall be added as a prefix to the line as described by the *ex* **autoindent** edit option.

<backslash>

Synopsis: <backslash>

Allow the entry of a subsequent <newline> or <control>-J as a literal character, removing any special meaning that it may have to the editor during text input mode. The <backslash> character shall be retained and evaluated when the command line is parsed, or retained and included when the input text becomes part of the edit buffer.

<control>-V

Synopsis: <control>-V

Allow the entry of any subsequent character as a literal character, removing any special meaning that it may have to the editor during text input mode. The <control>-V character shall be discarded before the command line is parsed or the input text becomes part of the edit buffer.

If the “literal next” functionality is performed by the underlying system, it is implementation-defined whether a character other than <control>-V performs this function.

<control>-W

Synopsis: <control>-W

Discard the <control>-W, and the word previous to it in the input line, including any <blank> characters following the word and preceding the <control>-W. If the “word erase” functionality is performed by the underlying system, it is implementation-defined whether a character other than <control>-W performs this function.

Command Descriptions in ex

The following symbols are used in this section to represent command modifiers. Some of these modifiers can be omitted, in which case the specified defaults shall be used.

laddr A single line address, given in any of the forms described in [Addressing in ex](#) (on page 2644); the default shall be the current line (‘.’), unless otherwise specified.

If the line address is zero, it shall be an error, unless otherwise specified in the following command descriptions.

ex*Utilities*

85776		If the edit buffer is empty, and the address is specified with a command other than
85777		= , append , insert , open , put , read , or visual , or the address is not zero, it shall be
85778		an error.
85779	<i>2addr</i>	Two addresses specifying an inclusive range of lines. If no addresses are specified,
85780		the default for <i>2addr</i> shall be the current line only (".,."), unless otherwise
85781		specified in the following command descriptions. If one address is specified, <i>2addr</i>
85782		shall specify that line only, unless otherwise specified in the following command
85783		descriptions.
85784		It shall be an error if the first address is greater than the second address.
85785		If the edit buffer is empty, and the two addresses are specified with a command
85786		other than the ! , write , wq , or xit commands, or either address is not zero, it shall
85787		be an error.
85788	<i>count</i>	A positive decimal number. If <i>count</i> is specified, it shall be equivalent to specifying
85789		an additional address to the command, unless otherwise specified by the following
85790		command descriptions. The additional address shall be equal to the last address
85791		specified to the command (either explicitly or by default) plus <i>count</i> –1.
85792		If this would result in an address greater than the last line of the edit buffer, it shall
85793		be corrected to equal the last line of the edit buffer.
85794	<i>flags</i>	One or more of the characters ' + ' , ' - ' , ' # ' , ' p ' , or ' l ' (ell). The flag characters
85795		can be <blank>-separated, and in any order or combination. The characters ' # ' ,
85796		' p ' , and ' l ' shall cause lines to be written in the format specified by the print
85797		command with the specified <i>flags</i> .
85798		The lines to be written are as follows:
85799		1. All edit buffer lines written during the execution of the <i>ex</i> & , ~ , list , number ,
85800		open , print , s , visual , and z commands shall be written as specified by <i>flags</i> .
85801		2. After the completion of an <i>ex</i> command with a flag as an argument, the
85802		current line shall be written as specified by <i>flags</i> , unless the current line was
85803		the last line written by the command.
85804		The characters ' + ' and ' - ' cause the value of the current line after the execution
85805		of the <i>ex</i> command to be adjusted by the offset address as described in Addressing
85806		in <i>ex</i> (on page 2644). This adjustment shall occur before the current line is written
85807		as described in 2. above.
85808		The default for <i>flags</i> shall be none.
85809	<i>buffer</i>	One of a number of named areas for holding text. The named buffers are specified
85810		by the alphanumeric characters of the POSIX locale. There shall also be one
85811		"unnamed" buffer. When no buffer is specified for editor commands that use a
85812		buffer, the unnamed buffer shall be used. Commands that store text into buffers
85813		shall store the text as it was before the command took effect, and shall store text
85814		occurring earlier in the file before text occurring later in the file, regardless of how
85815		the text region was specified. Commands that store text into buffers shall store the
85816		text into the unnamed buffer as well as any specified buffer.
85817		In <i>ex</i> commands, buffer names are specified as the name by itself. In open or visual
85818		mode commands the name is preceded by a double-quote (' ") character.
85819		If the specified buffer name is an uppercase character, and the buffer contents are
85820		to be modified, the buffer shall be appended to rather than being overwritten. If

85821		the buffer is not being modified, specifying the buffer name in lowercase and
85822		uppercase shall have identical results.
85823		There shall also be buffers named by the numbers 1 through 9. In open and visual
85824		mode, if a region of text including characters from more than a single line is being
85825		modified by the <i>vi</i> c or d commands, the motion character associated with the c or
85826		d commands specifies that the buffer text shall be in line mode, or the commands
85827		%, ', /, ?, (,), N, n, {, or } are used to define a region of text for the c or d commands,
85828		the contents of buffers 1 through 8 shall be moved into the buffer named by the
85829		next numerically greater value, the contents of buffer 9 shall be discarded, and the
85830		region of text shall be copied into buffer 1. This shall be in addition to copying the
85831		text into a user-specified buffer or unnamed buffer, or both. Numeric buffers can
85832		be specified as a source buffer for open and visual mode commands; however,
85833		specifying a numeric buffer as the write target of an open or visual mode
85834		command shall have unspecified results.
85835		The text of each buffer shall have the characteristic of being in either line or
85836		character mode. Appending text to a non-empty buffer shall set the mode to match
85837		the characteristic of the text being appended. Appending text to a buffer shall
85838		cause the creation of at least one additional line in the buffer. All text stored into
85839		buffers by <i>ex</i> commands shall be in line mode. The <i>ex</i> commands that use buffers
85840		as the source of text specify individually how buffers of different modes are
85841		handled. Each open or visual mode command that uses buffers for any purpose
85842		specifies individually the mode of the text stored into the buffer and how buffers
85843		of different modes are handled.
85844	<i>file</i>	Command text used to derive a pathname. The default shall be the current
85845		pathname, as defined previously, in which case, if no current pathname has yet
85846		been established it shall be an error, except where specifically noted in the
85847		individual command descriptions that follow. If the command text contains any of
85848		the characters '~', '{', '[', '*', '?', '\$', '"', backquote, single-quote, and
85849		<backslash>, it shall be subjected to the process of "shell expansions", as described
85850		below; if more than a single pathname results and the command expects only one,
85851		it shall be an error.
85852		The process of shell expansions in the editor shall be done as follows. The <i>ex</i> utility
85853		shall pass two arguments to the program named by the shell edit option; the first
85854		shall be -c, and the second shall be the string "echo" and the command text as a
85855		single argument. The standard output and standard error of that command shall
85856		replace the command text.
85857	!	A character that can be appended to the command name to modify its operation,
85858		as detailed in the individual command descriptions. With the exception of the <i>ex</i>
85859		read , write , and ! commands, the '!' character shall only act as a modifier if there
85860		are no <blank> characters between it and the command name.
85861	<i>remembered search direction</i>	
85862		The <i>vi</i> commands N and n begin searching in a forwards or backwards direction in
85863		the edit buffer based on a remembered search direction, which is initially unset,
85864		and is set by the <i>ex</i> global , v , s , and tag commands, and the <i>vi</i> / and ? commands.

Abbreviate

Synopsis: `ab[breviate] [lhs rhs]`

If *lhs* and *rhs* are not specified, write the current list of abbreviations and do nothing more.

Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable characters and <blank> characters shall not be restricted. Additional restrictions shall be implementation-defined.

In both *lhs* and *rhs*, any character may be escaped with a <control>-V, in which case the character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be discarded.

In open and visual text input mode, if a non-word or <ESC> character that is not escaped by a <control>-V character is entered after a word character, a check shall be made for a set of characters matching *lhs*, in the text input entered during this command. If it is found, the effect shall be as if *rhs* was entered instead of *lhs*.

The set of characters that are checked is defined as follows:

1. If there are no characters inserted before the word and non-word or <ESC> characters that triggered the check, the set of characters shall consist of the word character.
2. If the character inserted before the word and non-word or <ESC> characters that triggered the check is a word character, the set of characters shall consist of the characters inserted immediately before the triggering characters that are word characters, plus the triggering word character.
3. If the character inserted before the word and non-word or <ESC> characters that triggered the check is not a word character, the set of characters shall consist of the characters that were inserted before the triggering characters that are neither <blank> characters nor word characters, plus the triggering word character.

It is unspecified whether the *lhs* argument entered for the *ex* **abbreviate** and **unabbreviate** commands is replaced in this fashion. Regardless of whether or not the replacement occurs, the effect of the command shall be as if the replacement had not occurred.

Current line: Unchanged.

Current column: Unchanged.

Append

Synopsis: `[laddr] a[ppend] [!]`

Enter *ex* text input mode; the input text shall be placed after the specified line. If line zero is specified, the text shall be placed at the beginning of the edit buffer.

This command shall be affected by the **number** and **autoindent** edit options; following the command name with '!' shall cause the **autoindent** edit option setting to be toggled for the duration of this command only.

Current line: Set to the last input line; if no lines were input, set to the specified line, or to the first line of the edit buffer if a line of zero was specified, or zero if the edit buffer is empty.

Current column: Set to non-<blank>.

Arguments

Synopsis: ar[gs]

Write the current argument list, with the current argument-list entry, if any, between ' [' and '] ' characters.

Current line: Unchanged.

Current column: Unchanged.

Change

Synopsis: [2addr] c[hange][!][count]

Enter *ex* text input mode; the input text shall replace the specified lines. The specified lines shall be copied into the unnamed buffer, which shall become a line mode buffer.

This command shall be affected by the **number** and **autoindent** edit options; following the command name with ' ! ' shall cause the **autoindent** edit option setting to be toggled for the duration of this command only.

Current line: Set to the last input line; if no lines were input, set to the line before the first address, or to the first line of the edit buffer if there are no lines preceding the first address, or to zero if the edit buffer is empty.

Current column: Set to non-<blank>.

Change Directory

Synopsis: chd[ir][!][directory]
cd[!][directory]

Change the current working directory to *directory*.

If no *directory* argument is specified, and the *HOME* environment variable is set to a non-null and non-empty value, *directory* shall default to the value named in the *HOME* environment variable. If the *HOME* environment variable is empty or is undefined, the default value of *directory* is implementation-defined.

If no ' ! ' is appended to the command name, and the edit buffer has been modified since the last complete write, and the current pathname does not begin with a ' / ', it shall be an error.

Current line: Unchanged.

Current column: Unchanged.

Copy

Synopsis: [2addr] co[py] laddr [flags]
[2addr] t laddr [flags]

Copy the specified lines after the specified destination line; line zero specifies that the lines shall be placed at the beginning of the edit buffer.

Current line: Set to the last line copied.

Current column: Set to non-<blank>.

Delete

Synopsis: [2addr] d[ele]te[buffer][count][flags]

Delete the specified lines into a buffer (defaulting to the unnamed buffer), which shall become a line-mode buffer.

Flags can immediately follow the command name; see [Command Line Parsing in ex](#) (on page 2646).

Current line: Set to the line following the deleted lines, or to the last line in the edit buffer if that line is past the end of the edit buffer, or to zero if the edit buffer is empty.

Current column: Set to non-<blank>.

Edit

Synopsis: e[dit][!][+command][file]
ex[!][+command][file]

If no '!' is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error.

If *file* is specified, replace the current contents of the edit buffer with the current contents of *file*, and set the current pathname to *file*. If *file* is not specified, replace the current contents of the edit buffer with the current contents of the file named by the current pathname. If for any reason the current contents of the file cannot be accessed, the edit buffer shall be empty.

The *+command* option shall be <blank>-delimited; <blank> characters within the *+command* can be escaped by preceding them with a <backslash> character. The *+command* shall be interpreted as an *ex* command immediately after the contents of the edit buffer have been replaced and the current line and column have been set.

If the edit buffer is empty:

Current line: Set to 0.

Current column: Set to 1.

Otherwise, if executed while in *ex* command mode or if the *+command* argument is specified:

Current line: Set to the last line of the edit buffer.

Current column: Set to non-<blank>.

Otherwise, if *file* is omitted or results in the current pathname:

Current line: Set to the first line of the edit buffer.

Current column: Set to non-<blank>.

Otherwise, if *file* is the same as the last file edited, the line and column shall be set as follows; if the file was previously edited, the line and column may be set as follows:

Current line: Set to the last value held when that file was last edited. If this value is not a valid line in the new edit buffer, set to the first line of the edit buffer.

Current column: If the current line was set to the last value held when the file was last edited, set to the last value held when the file was last edited. Otherwise, or if the last value is not a valid column in the new edit buffer, set to non-<blank>.

Otherwise:

Current line: Set to the first line of the edit buffer.

Current column: Set to non-<blank>.

File

Synopsis: f[ile] [file]

If a *file* argument is specified, the alternate pathname shall be set to the current pathname, and the current pathname shall be set to *file*.

Write an informational message. If the file has a current pathname, it shall be included in this message; otherwise, the message shall indicate that there is no current pathname. If the edit buffer contains lines, the current line number and the number of lines in the edit buffer shall be included in this message; otherwise, the message shall indicate that the edit buffer is empty. If the edit buffer has been modified since the last complete write, this fact shall be included in this message. If the **readonly** edit option is set, this fact shall be included in this message. The message may contain other unspecified information.

Current line: Unchanged.

Current column: Unchanged.

Global

Synopsis: [2addr] g[lobal] /pattern/ [commands]
 [2addr] v /pattern/ [commands]

The optional **'!'** character after the **global** command shall be the same as executing the **v** command.

If *pattern* is empty (for example, **"v"**) or not specified, the last regular expression used in the editor command shall be used as the *pattern*. The *pattern* can be delimited by <slash> characters (shown in the Synopsis), as well as any non-alphanumeric or non-<blank> other than <backslash>, <vertical-line>, <newline>, or double-quote.

If no lines are specified, the lines shall default to the entire file.

The **global** and **v** commands are logically two-pass operations. First, mark the lines within the specified lines for which the line excluding the terminating <newline> matches (**global**) or does not match (**v** or **global!**) the specified pattern. Second, execute the *ex* commands given by *commands*, with the current line (**'.'**) set to each marked line. If an error occurs during this process, or the contents of the edit buffer are replaced (for example, by the *ex* **:edit** command) an error message shall be written and no more commands resulting from the execution of this command shall be processed.

Multiple *ex* commands can be specified by entering multiple commands on a single line using a <vertical-line> to delimit them, or one per line, by escaping each <newline> with a <backslash>.

If no commands are specified:

1. If in *ex* command mode, it shall be as if the **print** command were specified.
2. Otherwise, no command shall be executed.

For the **append**, **change**, and **insert** commands, the input text shall be included as part of the command, and the terminating <period> can be omitted if the command ends the list of commands. The **open** and **visual** commands can be specified as one of the commands, in which

86018 case each marked line shall cause the editor to enter open or visual mode. If open or visual mode
 86019 is exited using the *vi* **Q** command, the current line shall be set to the next marked line, and open
 86020 or visual mode reentered, until the list of marked lines is exhausted.

86021 The **global**, **v**, and **undo** commands cannot be used in *commands*. Marked lines may be deleted
 86022 by commands executed for lines occurring earlier in the file than the marked lines. In this case,
 86023 no commands shall be executed for the deleted lines.

86024 If the remembered search direction is not set, the **global** and **v** commands shall set it to forward.

86025 The **autoprint** and **autoindent** edit options shall be inhibited for the duration of the **g** or **v**
 86026 command.

86027 *Current line*: If no commands executed, set to the last marked line. Otherwise, as specified for the
 86028 executed *ex* commands.

86029 *Current column*: If no commands are executed, set to non-*<blank>*; otherwise, as specified for the
 86030 individual *ex* commands.

86031 Insert

86032 *Synopsis*: `[1addr] i[nsert][!]`

86033 Enter *ex* text input mode; the input text shall be placed before the specified line. If the line is zero
 86034 or 1, the text shall be placed at the beginning of the edit buffer.

86035 This command shall be affected by the **number** and **autoindent** edit options; following the
 86036 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
 86037 duration of this command only.

86038 *Current line*: Set to the last input line; if no lines were input, set to the line before the specified
 86039 line, or to the first line of the edit buffer if there are no lines preceding the specified line, or zero
 86040 if the edit buffer is empty.

86041 *Current column*: Set to non-*<blank>*.

86042 Join

86043 *Synopsis*: `[2addr] j[oin][!][count][flags]`

86044 If *count* is specified:

86045 If no address was specified, the **join** command shall behave as if *2addr* were the current
 86046 line and the current line plus *count* (*.,. + count*).

86047 If one address was specified, the **join** command shall behave as if *2addr* were the specified
 86048 address and the specified address plus *count* (*addr,addr + count*).

86049 If two addresses were specified, the **join** command shall behave as if an additional
 86050 address, equal to the last address plus *count* - 1 (*addr1,addr2,addr2 + count - 1*), was
 86051 specified.

86052 If this would result in a second address greater than the last line of the edit buffer, it shall
 86053 be corrected to be equal to the last line of the edit buffer.

86054 If no *count* is specified:

86055 If no address was specified, the **join** command shall behave as if *2addr* were the current
 86056 line and the next line (*.*, *.* +1).

86057 If one address was specified, the **join** command shall behave as if *2addr* were the specified
 86058 address and the next line (*addr*, *addr* +1).

86059 Join the text from the specified lines together into a single line, which shall replace the specified
 86060 lines.

86061 If a '!' character is appended to the command name, the **join** shall be without modification of
 86062 any line, independent of the current locale.

86063 Otherwise, in the POSIX locale, set the current line to the first of the specified lines, and then, for
 86064 each subsequent line, proceed as follows:

- 86065 1. Discard leading <space> characters from the line to be joined.
- 86066 2. If the line to be joined is now empty, delete it, and skip steps 3 through 5.
- 86067 3. If the current line ends in a <blank>, or the first character of the line to be joined is a ') ' character, join the lines without further modification.
- 86068 4. If the last character of the current line is a ' . ' , join the lines with two <space> characters between them.
- 86069 5. Otherwise, join the lines with a single <space> between them.

86072 *Current line*: Set to the first line specified.

86073 *Current column*: Set to non-<blank>.

86074 List

86075 *Synopsis*: `[2addr] l[ist][count][flags]`

86076 This command shall be equivalent to the *ex* command:

86077 `[2addr] p[rint][count] l[flags]`

86078 See [Print](#) (on page 2663).

86079 Map

86080 *Synopsis*: `map[!][lhs rhs]`

86081 If *lhs* and *rhs* are not specified:

- 86082 1. If '!' is specified, write the current list of text input mode maps.
- 86083 2. Otherwise, write the current list of command mode maps.
- 86084 3. Do nothing more.

86085 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable
 86086 characters and <blank> characters shall not be restricted. Additional restrictions shall be
 86087 implementation-defined. In both *lhs* and *rhs*, any character can be escaped with a <control>-V, in
 86088 which case the character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V
 86089 shall be discarded.

86090 If the character '!' is appended to the **map** command name, the mapping shall be effective
 86091 during open or visual text input mode rather than **open** or **visual** command mode. This allows
 86092 *lhs* to have two different **map** definitions at the same time: one for command mode and one for

- 86093 text input mode.
- 86094 For command mode mappings:
- 86095 When the *lhs* is entered as any part of a *vi* command in open or visual mode (but not as
- 86096 part of the arguments to the command), the action shall be as if the corresponding *rhs* had
- 86097 been entered.
- 86098 If any character in the command, other than the first, is escaped using a <control>-V
- 86099 character, that character shall not be part of a match to an *lhs*.
- 86100 It is unspecified whether implementations shall support **map** commands where the *lhs* is
- 86101 more than a single character in length, where the first character of the *lhs* is printable.
- 86102 If *lhs* contains more than one character and the first character is '#', followed by a
- 86103 sequence of digits corresponding to a numbered function key, then when this function key
- 86104 is typed it shall be mapped to *rhs*. Characters other than digits following a '#' character
- 86105 also represent the function key named by the characters in the *lhs* following the '#' and
- 86106 may be mapped to *rhs*. It is unspecified how function keys are named or what function
- 86107 keys are supported.
- 86108 For text input mode mappings:
- 86109 When the *lhs* is entered as any part of text entered in open or visual text input modes, the
- 86110 action shall be as if the corresponding *rhs* had been entered.
- 86111 If any character in the input text is escaped using a <control>-V character, that character
- 86112 shall not be part of a match to an *lhs*.
- 86113 It is unspecified whether the *lhs* text entered for subsequent **map** or **unmap** commands is
- 86114 replaced with the *rhs* text for the purposes of the screen display; regardless of whether or
- 86115 not the display appears as if the corresponding *rhs* text was entered, the effect of the
- 86116 command shall be as if the *lhs* text was entered.
- 86117 If only part of the *lhs* is entered, it is unspecified how long the editor will wait for additional,
- 86118 possibly matching characters before treating the already entered characters as not matching the
- 86119 *lhs*.
- 86120 The *rhs* characters shall themselves be subject to remapping, unless otherwise specified by the
- 86121 **remap** edit option, except that if the characters in *lhs* occur as prefix characters in *rhs*, those
- 86122 characters shall not be remapped.
- 86123 On block-mode terminals, the mapping need not occur immediately (for example, it may occur
- 86124 after the terminal transmits a group of characters to the system), but it shall achieve the same
- 86125 results as if it occurred immediately.
- 86126 *Current line*: Unchanged.
- 86127 *Current column*: Unchanged.

Mark

Synopsis: [*laddr*] ma[*rk*] *character*

 [*laddr*] k *character*

Implementations shall support *character* values of a single lowercase letter of the POSIX locale and the backquote and single-quote characters; support of other characters is implementation-defined.

If executing the *vi m* command, set the specified mark to the current line and 1-based numbered character referenced by the current column, if any; otherwise, column position 1.

Otherwise, set the specified mark to the specified line and 1-based numbered first non-<blank> non-<newline> in the line, if any; otherwise, the last non-<newline> in the line, if any; otherwise, column position 1.

The mark shall remain associated with the line until the mark is reset or the line is deleted. If a deleted line is restored by a subsequent **undo** command, any marks previously associated with the line, which have not been reset, shall be restored as well. Any use of a mark not associated with a current line in the edit buffer shall be an error.

The marks ' and ' shall be set as described previously, immediately before the following events occur in the editor:

1. The use of ' \$ ' as an *ex* address
2. The use of a positive decimal number as an *ex* address
3. The use of a search command as an *ex* address
4. The use of a mark reference as an *ex* address
5. The use of the following open and visual mode commands: <control>-, %, (,), [,], {, }
6. The use of the following open and visual mode commands: ', G, H, L, M, z if the current line will change as a result of the command
7. The use of the open and visual mode commands: /, ?, N, ', n if the current line or column will change as a result of the command
8. The use of the *ex* mode commands: z, **undo**, **global**, v

For rules 1., 2., 3., and 4., the ' and ' marks shall not be set if the *ex* command is parsed as specified by rule 6.a. in [Command Line Parsing in ex](#) (on page 2646).

For rules 5., 6., and 7., the ' and ' marks shall not be set if the commands are used as motion commands in open and visual mode.

For rules 1., 2., 3., 4., 5., 6., 7., and 8., the ' and ' marks shall not be set if the command fails.

The ' and ' marks shall be set as described previously, each time the contents of the edit buffer are replaced (including the editing of the initial buffer), if in open or visual mode, or if in **ex** mode and the edit buffer is not empty, before any commands or movements (including commands or movements specified by the -c or -t options or the +*command* argument) are executed on the edit buffer. If in open or visual mode, the marks shall be set as if executing the *vi m* command; otherwise, as if executing the *ex mark* command.

When changing from **ex** mode to open or visual mode, if the ' and ' marks are not already set, the ' and ' marks shall be set as described previously.

Current line: Unchanged.

86169 *Current column*: Unchanged.

86170 **Move**

86171 *Synopsis*: [2addr] m[ove] laddr [flags]

86172 Move the specified lines after the specified destination line. A destination of line zero specifies
86173 that the lines shall be placed at the beginning of the edit buffer. It shall be an error if the
86174 destination line is within the range of lines to be moved.

86175 *Current line*: Set to the last of the moved lines.

86176 *Current column*: Set to non-<blank>.

86177 **Next**

86178 *Synopsis*: n[ext] [!] [+command] [file ...]

86179 If no '!' is appended to the command name, and the edit buffer has been modified since the
86180 last complete write, it shall be an error, unless the file is successfully written as specified by the
86181 **autowrite** option.

86182 If one or more files is specified:

- 86183 1. Set the argument list to the specified filenames.
- 86184 2. Set the current argument list reference to be the first entry in the argument list.
- 86185 3. Set the current pathname to the first filename specified.

86186 Otherwise:

- 86187 1. It shall be an error if there are no more filenames in the argument list after the filename
86188 currently referenced.
- 86189 2. Set the current pathname and the current argument list reference to the filename after the
86190 filename currently referenced in the argument list.

86191 Replace the contents of the edit buffer with the contents of the file named by the current
86192 pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be
86193 empty.

86194 This command shall be affected by the **autowrite** and **writeany** edit options.

86195 The *+command* option shall be <blank>-delimited; <blank> characters can be escaped by
86196 preceding them with a <backslash> character. The *+command* shall be interpreted as an *ex*
86197 command immediately after the contents of the edit buffer have been replaced and the current
86198 line and column have been set.

86199 *Current line*: Set as described for the **edit** command.

86200 *Current column*: Set as described for the **edit** command.

Number

Synopsis: [2addr] nu[mber] [count] [flags]
 [2addr] # [count] [flags]

These commands shall be equivalent to the *ex* command:

[2addr] p[rint] [count] # [flags]

See [Print](#).

Open

Synopsis: [1addr] o[pen] /pattern/ [flags]

This command need not be supported on block-mode terminals or terminals with insufficient capabilities. If standard input, standard output, or standard error are not terminal devices, the results are unspecified.

Enter open mode.

The trailing delimiter can be omitted from *pattern* at the end of the command line. If *pattern* is empty (for example, "//") or not specified, the last regular expression used in the editor shall be used as the pattern. The pattern can be delimited by <slash> characters (shown in the Synopsis), as well as any alphanumeric, or non-<blank> other than <backslash>, <vertical-line>, <newline>, or double-quote.

Current line: Set to the specified line.

Current column: Set to non-<blank>.

Preserve

Synopsis: pre[serve]

Save the edit buffer in a form that can later be recovered by using the *-r* option or by using the *ex recover* command. After the file has been preserved, a mail message shall be sent to the user. This message shall be readable by invoking the *mailx* utility. The message shall contain the name of the file, the time of preservation, and an *ex* command that could be used to recover the file. Additional information may be included in the mail message.

Current line: Unchanged.

Current column: Unchanged.

Print

Synopsis: [2addr] p[rint] [count] [flags]

Write the addressed lines. The behavior is unspecified if the number of columns on the display is less than the number of columns required to write any single character in the lines being written.

Non-printable characters, except for the <tab>, shall be written as implementation-defined multi-character sequences.

If the # flag is specified or the **number** edit option is set, each line shall be preceded by its line number in the following format:

"%6dΔΔ", <line number>

If the l flag is specified or the **list** edit option is set:

1. The characters listed in XBD Table 5-1 (on page 121) shall be written as the corresponding escape sequence.
2. Non-printable characters not in XBD Table 5-1 (on page 121) shall be written as one three-digit octal number (with a preceding <backslash>) for each byte in the character (most significant byte first).
3. The end of each line shall be marked with a '\$', and literal '\$' characters within the line shall be written with a preceding <backslash>.

Long lines shall be folded; the length at which folding occurs is unspecified, but should be appropriate for the output terminal, considering the number of columns of the terminal.

If a line is folded, and the **l** flag is not specified and the **list** edit option is not set, it is unspecified whether a multi-column character at the folding position is separated; it shall not be discarded.

Current line: Set to the last written line.

Current column: Unchanged if the current line is unchanged; otherwise, set to non-<blank>.

Put

Synopsis: [*laddr*] pu[t][*buffer*]

Append text from the specified buffer (by default, the unnamed buffer) to the specified line; line zero specifies that the text shall be placed at the beginning of the edit buffer. Each portion of a line in the buffer shall become a new line in the edit buffer, regardless of the mode of the buffer.

Current line: Set to the last line entered into the edit buffer.

Current column: Set to non-<blank>.

Quit

Synopsis: q[uit][!]

If no '!' is appended to the command name:

1. If the edit buffer has been modified since the last complete write, it shall be an error.
2. If there are filenames in the argument list after the filename currently referenced, and the last command was not a **quit**, **wq**, **xit**, or **ZZ** (see **Exit**, on page 3344) command, it shall be an error.

Otherwise, terminate the editing session.

Read

Synopsis: [*laddr*] r[ead][!][*file*]

If '!' is not the first non-<blank> to follow the command name, a copy of the specified file shall be appended into the edit buffer after the specified line; line zero specifies that the copy shall be placed at the beginning of the edit buffer. The number of lines and bytes read shall be written. If no *file* is named, the current pathname shall be the default. If there is no current pathname, then *file* shall become the current pathname. If there is no current pathname or *file* operand, it shall be an error. Specifying a *file* that is not of type regular shall have unspecified results.

Otherwise, if *file* is preceded by '!', the rest of the line after the '!' shall have '%', '#', and '!' characters expanded as described in **Command Line Parsing in ex** (on page 2646).

The **ex** utility shall then pass two arguments to the program named by the shell edit option; the

first shall be `-c` and the second shall be the expanded arguments to the **read** command as a single argument. The standard input of the program shall be set to the standard input of the *ex* program when it was invoked. The standard error and standard output of the program shall be appended into the edit buffer after the specified line.

Each line in the copied file or program output (as delimited by `<newline>` characters or the end of the file or output if it is not immediately preceded by a `<newline>`), shall be a separate line in the edit buffer. Any occurrences of `<carriage-return>` and `<newline>` pairs in the output shall be treated as single `<newline>` characters.

The special meaning of the `' ! '` following the **read** command can be overridden by escaping it with a `<backslash>` character.

Current line: If no lines are added to the edit buffer, unchanged. Otherwise, if in open or visual mode, set to the first line entered into the edit buffer. Otherwise, set to the last line entered into the edit buffer.

Current column: Set to non-`<blank>`.

Recover

Synopsis: `rec[over] [!] file`

If no `' ! '` is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error.

If no *file* operand is specified, then the current pathname shall be used. If there is no current pathname or *file* operand, it shall be an error.

If no recovery information has previously been saved about *file*, the **recover** command shall behave identically to the **edit** command, and an informational message to this effect shall be written.

Otherwise, set the current pathname to *file*, and replace the current contents of the edit buffer with the recovered contents of *file*. If there are multiple instances of the file to be recovered, the one most recently saved shall be recovered, and an informational message that there are previous versions of the file that can be recovered shall be written. The editor shall behave as if the contents of the edit buffer have already been modified.

Current file: Set as described for the **edit** command.

Current column: Set as described for the **edit** command.

Rewind

Synopsis: `rew[ind] [!]`

If no `' ! '` is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error, unless the file is successfully written as specified by the **autowrite** option.

If the argument list is empty, it shall be an error.

The current argument list reference and the current pathname shall be set to the first filename in the argument list.

Replace the contents of the edit buffer with the contents of the file named by the current pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be empty.

This command shall be affected by the **autowrite** and **writany** edit options.

Current line: Set as described for the **edit** command.

Current column: Set as described for the **edit** command.

Set

Synopsis: `se[t][option]=[value] ...][nooption ...][option? ...][all]`

When no arguments are specified, write the value of the **term** edit option and those options whose values have been changed from the default settings; when the argument *all* is specified, write all of the option values.

Giving an option name followed by the character '?' shall cause the current value of that option to be written. The '?' can be separated from the option name by zero or more <blank> characters. The '?' shall be necessary only for Boolean valued options. Boolean options can be given values by the form **set option** to turn them on or **set nooption** to turn them off; string and numeric options can be assigned by the form **set option=value**. Any <blank> characters in strings can be included as is by preceding each <blank> with an escaping <backslash>. More than one option can be set or listed by a single set command by specifying multiple arguments, each separated from the next by one or more <blank> characters.

See [Edit Options in ex](#) (on page 2676) for details about specific options.

Current line: Unchanged.

Current column: Unchanged.

Shell

Synopsis: `sh[ell]`

Invoke the program named in the **shell** edit option with the single argument **-i** (interactive mode). Editing shall be resumed when the program exits.

Current line: Unchanged.

Current column: Unchanged.

Source

Synopsis: `so[urce] file`

Read and execute *ex* commands from *file*. Lines in the file that are blank lines shall be ignored.

Current line: As specified for the individual *ex* commands.

Current column: As specified for the individual *ex* commands.

Substitute

Synopsis: `[2addr] s[substitute][/pattern/repl][options][count][flags]]`
 `[2addr] &[options][count][flags]]`
 `[2addr] ~[options][count][flags]]`

Replace the first instance of the pattern *pattern* by the string *repl* on each specified line. (See [Regular Expressions in ex](#) (on page 2675) and [Replacement Strings in ex](#) (on page 2676).) Any non-alphabetic, non-<blank> delimiter other than <backslash>, '&|', <newline>, or double-quote can be used instead of '/'. <backslash> characters can be used to escape delimiters, <backslash> characters, and other special characters.

The trailing delimiter can be omitted from *pattern* or from *repl* at the end of the command line. If both *pattern* and *repl* are not specified or are empty (for example, `//`), the last `s` command shall be repeated. If only *pattern* is not specified or is empty, the last regular expression used in the editor shall be used as the pattern. If only *repl* is not specified or is empty, the pattern shall be replaced by nothing. If the entire replacement pattern is `' % '`, the last replacement pattern to an `s` command shall be used.

Entering a <carriage-return> in *repl* (which requires an escaping <backslash> in *ex* mode and an escaping <control>-V in open or *vi* mode) shall split the line at that point, creating a new line in the edit buffer. The <carriage-return> shall be discarded.

If *options* includes the letter `' g '` (**global**), all non-overlapping instances of the pattern in the line shall be replaced.

If *options* includes the letter `' c '` (**confirm**), then before each substitution the line shall be written; the written line shall reflect all previous substitutions. On the following line, <space> characters shall be written beneath the characters from the line that are before the *pattern* to be replaced, and `' ^ '` characters written beneath the characters included in the *pattern* to be replaced. The *ex* utility shall then wait for a response from the user. An affirmative response shall cause the substitution to be done, while any other input shall not make the substitution. An affirmative response shall consist of a line with the affirmative response (as defined by the current locale) at the beginning of the line. This line shall be subject to editing in the same way as the *ex* command line.

If interrupted (see the ASYNCHRONOUS EVENTS section), any modifications confirmed by the user shall be preserved in the edit buffer after the interrupt.

If the remembered search direction is not set, the `s` command shall set it to forward.

In the second Synopsis, the `&` command shall repeat the previous substitution, as if the `&` command were replaced by:

```
s/pattern/repl/
```

where *pattern* and *repl* are as specified in the previous `s`, `&`, or `~` command.

In the third Synopsis, the `~` command shall repeat the previous substitution, as if the `' ~ '` were replaced by:

```
s/pattern/repl
```

where *pattern* shall be the last regular expression specified to the editor, and *repl* shall be from the previous substitution (including `&` and `~`) command.

These commands shall be affected by the `LC_MESSAGES` environment variable.

Current line: Set to the last line in which a substitution occurred, or, unchanged if no substitution occurred.

Current column: Set to non-<blank>.

Suspend

Synopsis: su[suspend] [!]
 st[op] [!]

Allow control to return to the invoking process; *ex* shall suspend itself as if it had received the SIGTSTP signal. The suspension shall occur only if job control is enabled in the invoking shell (see the description of *set -m*).

These commands shall be affected by the **autowrite** and **writeany** edit options.

The current **susp** character (see *stty*) shall be equivalent to the **suspend** command.

Tag

Synopsis: ta[g] [!] tagstring

The results are unspecified if the format of a tags file is not as specified by the *ctags* utility (see *ctags*) description.

The **tag** command shall search for *tagstring* in the tag files referred to by the **tag** edit option, in the order they are specified, until a reference to *tagstring* is found. Files shall be searched from beginning to end. If no reference is found, it shall be an error and an error message to this effect shall be written. If the reference is not found, or if an error occurs while processing a file referred to in the **tag** edit option, it shall be an error, and an error message shall be written at the first occurrence of such an error.

Otherwise, if the tags file contained a pattern, the pattern shall be treated as a regular expression used in the editor; for example, for the purposes of the **s** command.

If the *tagstring* is in a file with a different name than the current pathname, set the current pathname to the name of that file, and replace the contents of the edit buffer with the contents of that file. In this case, if no '!' is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error, unless the file is successfully written as specified by the **autowrite** option.

This command shall be affected by the **autowrite**, **tag**, **taglength**, and **writeany** edit options.

Current line: If the tags file contained a line number, set to that line number. If the line number is larger than the last line in the edit buffer, an error message shall be written and the current line shall be set as specified for the **edit** command.

If the tags file contained a pattern, set to the first occurrence of the pattern in the file. If no matching pattern is found, an error message shall be written and the current line shall be set as specified for the **edit** command.

Current column: If the tags file contained a line-number reference and that line-number was not larger than the last line in the edit buffer, or if the tags file contained a pattern and that pattern was found, set to non-<blank>. Otherwise, set as specified for the **edit** command.

Unabbreviate

Synopsis: una[bbrev] lhs

If *lhs* is not an entry in the current list of abbreviations (see [Abbreviate](#), on page 2654), it shall be an error. Otherwise, delete *lhs* from the list of abbreviations.

Current line: Unchanged.

Current column: Unchanged.

Undo

Synopsis: u[ndo]

Reverse the changes made by the last command that modified the contents of the edit buffer, including **undo**. For this purpose, the **global**, **v**, **open**, and **visual** commands, and commands resulting from buffer executions and mapped character expansions, are considered single commands.

If no action that can be undone preceded the **undo** command, it shall be an error.

If the **undo** command restores lines that were marked, the mark shall also be restored unless it was reset subsequent to the deletion of the lines.

Current line:

1. If lines are added or changed in the file, set to the first line added or changed.
2. Set to the line before the first line deleted, if it exists.
3. Set to 1 if the edit buffer is not empty.
4. Set to zero.

Current column: Set to non-<blank>.

Unmap

Synopsis: unm[ap] [!:] lhs

If '!' is appended to the command name, and if *lhs* is not an entry in the list of text input mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of text input mode map definitions.

If no '!' is appended to the command name, and if *lhs* is not an entry in the list of command mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of command mode map definitions.

Current line: Unchanged.

Current column: Unchanged.

Version

Synopsis: ve[rsion]

Write a message containing version information for the editor. The format of the message is unspecified.

Current line: Unchanged.

Current column: Unchanged.

Visual

Synopsis: [laddr] vi[sual][type][count][flags]

If *ex* is currently in open or visual mode, the Synopsis and behavior of the visual command shall be the same as the **edit** command, as specified by **Edit** (on page 2656).

Otherwise, this command need not be supported on block-mode terminals or terminals with insufficient capabilities. If standard input, standard output, or standard error are not terminal devices, the results are unspecified.

If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in **window**, on page 2683). If the '^' type character was also specified, the **window** edit option shall be set before being used by the type character.

Enter visual mode. If *type* is not specified, it shall be as if a *type* of '+' was specified. The *type* shall cause the following effects:

- + Place the beginning of the specified line at the top of the display.
- Place the end of the specified line at the bottom of the display.
- . Place the beginning of the specified line in the middle of the display.
- ^ If the specified line is less than or equal to the value of the **window** edit option, set the line to 1; otherwise, decrement the line by the value of the **window** edit option minus 1. Place the beginning of this line as close to the bottom of the displayed lines as possible, while still displaying the value of the **window** edit option number of lines.

Current line: Set to the specified line.

Current column: Set to non-<blank>.

Write

Synopsis: [2addr] w[rite][!][>>][file]
 [2addr] w[rite][!][file]
 [2addr] wq[!][>>][file]

If no lines are specified, the lines shall default to the entire file.

The command **wq** shall be equivalent to a **write** command followed by a **quit** command; **wq!** shall be equivalent to **write!** followed by **quit**. In both cases, if the **write** command fails, the **quit** shall not be attempted.

If the command name is not followed by one or more <blank> characters, or *file* is not preceded by a '!' character, the **write** shall be to a file.

1. If the >> argument is specified, and the file already exists, the lines shall be appended to the file instead of replacing its contents. If the >> argument is specified, and the file does not already exist, it is unspecified whether the write shall proceed as if the >> argument

- 86500 had not been specified or if the write shall fail.
- 86501 2. If the **readonly** edit option is set (see **readonly**, on page 2680), the **write** shall fail.
- 86502 3. If *file* is specified, and is not the current pathname, and the file exists, the **write** shall fail.
- 86503 4. If *file* is not specified, the current pathname shall be used. If there is no current pathname,
- 86504 the **write** command shall fail.
- 86505 5. If the current pathname is used, and the current pathname has been changed by the **file**
- 86506 or **read** commands, and the file exists, the **write** shall fail. If the **write** is successful,
- 86507 subsequent **writes** shall not fail for this reason (unless the current pathname is changed
- 86508 again).
- 86509 6. If the whole edit buffer is not being written, and the file to be written exists, the **write**
- 86510 shall fail.
- 86511 For rules 1., 2., 3., and 5., the **write** can be forced by appending the character '!' to the
- 86512 command name.
- 86513 For rules 2., 3., and 5., the **write** can be forced by setting the **writenany** edit option.
- 86514 Additional, implementation-defined tests may cause the **write** to fail.
- 86515 If the edit buffer is empty, a file without any contents shall be written.
- 86516 An informational message shall be written noting the number of lines and bytes written.
- 86517 Otherwise, if the command is followed by one or more <blank> characters, and the file is
- 86518 preceded by '!', the rest of the line after the '!' shall have '%', '#', and '!' characters
- 86519 expanded as described in **Command Line Parsing in ex** (on page 2646).
- 86520 The *ex* utility shall then pass two arguments to the program named by the **shell** edit option; the
- 86521 first shall be **-c** and the second shall be the expanded arguments to the **write** command as a
- 86522 single argument. The specified lines shall be written to the standard input of the command. The
- 86523 standard error and standard output of the program, if any, shall be written as described for the
- 86524 **print** command. If the last character in that output is not a <newline>, a <newline> shall be
- 86525 written at the end of the output.
- 86526 The special meaning of the '!' following the **write** command can be overridden by escaping it
- 86527 with a <backslash> character.
- 86528 *Current line*: Unchanged.
- 86529 *Current column*: Unchanged.
- 86530 **Write and Exit**
- 86531 *Synopsis:* [2addr] x[it][!][file]
- 86532 If the edit buffer has not been modified since the last complete **write**, **xit** shall be equivalent to
- 86533 the **quit** command, or if a '!' is appended to the command name, to **quit!**.
- 86534 Otherwise, **xit** shall be equivalent to the **wq** command, or if a '!' is appended to the command
- 86535 name, to **wq!**.
- 86536 *Current line*: Unchanged.
- 86537 *Current column*: Unchanged.

Yank

Synopsis: [*laddr*] ya[*nk*][*buffer*][*count*]

Copy the specified lines to the specified buffer (by default, the unnamed buffer), which shall become a line-mode buffer.

Current line: Unchanged.

Current column: Unchanged.

Adjust Window

Synopsis: [*laddr*] z[!][*type* ...][*count*][*flags*]

If no line is specified, the current line shall be the default; if *type* is omitted as well, the current line value shall first be incremented by 1. If incrementing the current line would cause it to be greater than the last line in the edit buffer, it shall be an error.

If there are <blank> characters between the *type* argument and the preceding **z** command name or optional '!' character, it shall be an error.

If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in **window**, on page 2683). If *count* is omitted, it shall default to 2 times the value of the **scroll** edit option, or if ! was specified, the number of lines in the display minus 1.

If *type* is omitted, then *count* lines starting with the specified line shall be written. Otherwise, *count* lines starting with the line specified by the *type* argument shall be written.

The *type* argument shall change the lines to be written. The possible values of *type* are as follows:

- The specified line shall be decremented by the following value:

$((\text{number of ``-'' characters}) \times \text{count}) - 1$

If the calculation would result in a number less than 1, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

- + The specified line shall be incremented by the following value:

$((\text{number of ``+'' characters}) - 1) \times \text{count} + 1$

If the calculation would result in a number greater than the last line in the edit buffer, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

=,. If more than a single '.' or '=' is specified, it shall be an error. The following steps shall be taken:

1. If *count* is zero, nothing shall be written.
2. Write as many of the *N* lines before the current line in the edit buffer as exist. If *count* or '!' was specified, *N* shall be:

$(\text{count} - 1) / 2$

Otherwise, *N* shall be:

$(\text{count} - 3) / 2$

If *N* is a number less than 3, no lines shall be written.

3. If '=' was specified as the type character, write a line consisting of the smaller of the number of columns in the display divided by two, or 40 '-' characters.
4. Write the current line.
5. Repeat step 3.
6. Write as many of the *N* lines after the current line in the edit buffer as exist. *N* shall be defined as in step 2. If *N* is a number less than 3, no lines shall be written. If *count* is less than 3, no lines shall be written.

^ The specified line shall be decremented by the following value:

```
((number of '^' characters) + 1) x count - 1
```

If the calculation would result in a number less than 1, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

Current line: Set to the last line written, unless the type is =, in which case, set to the specified line.

Current column: Set to non-<blank>.

Escape

Synopsis: ! *command*
 [*addr*] ! *command*

The contents of the line after the '!' shall have '%', '#', and '!' characters expanded as described in [Command Line Parsing in ex](#) (on page 2646). If the expansion causes the text of the line to change, it shall be redisplayed, preceded by a single '!' character.

The *ex* utility shall execute the program named by the **shell** edit option. It shall pass two arguments to the program; the first shall be -c, and the second shall be the expanded arguments to the ! command as a single argument.

If no lines are specified, the standard input, standard output, and standard error of the program shall be set to the standard input, standard output, and standard error of the *ex* program when it was invoked. In addition, a warning message shall be written if the edit buffer has been modified since the last complete write, and the **warn** edit option is set.

If lines are specified, they shall be passed to the program as standard input, and the standard output and standard error of the program shall replace those lines in the edit buffer. Each line in the program output (as delimited by <newline> characters or the end of the output if it is not immediately preceded by a <newline>), shall be a separate line in the edit buffer. Any occurrences of <carriage-return> and <newline> pairs in the output shall be treated as single <newline> characters. The specified lines shall be copied into the unnamed buffer before they are replaced, and the unnamed buffer shall become a line-mode buffer.

If in *ex* mode, a single '!' character shall be written when the program completes.

This command shall be affected by the **shell** and **warn** edit options. If no lines are specified, this command shall be affected by the **autowrite** and **writeany** edit options. If lines are specified, this command shall be affected by the **autoprint** edit option.

86615 *Current line:*

- 86616 1. If no lines are specified, unchanged.
- 86617 2. Otherwise, set to the last line read in, if any lines are read in.
- 86618 3. Otherwise, set to the line before the first line of the lines specified, if that line exists.
- 86619 4. Otherwise, set to the first line of the edit buffer if the edit buffer is not empty.
- 86620 5. Otherwise, set to zero.

86621 *Current column:* If no lines are specified, unchanged. Otherwise, set to non-<blank>.

86622 **Shift Left**

86623 *Synopsis:* **[2addr] <[< ...] [count] [flags]**

86624 Shift the specified lines to the start of the line; the number of column positions to be shifted shall
86625 be the number of command characters times the value of the **shiftwidth** edit option. Only
86626 leading <blank> characters shall be deleted or changed into other <blank> characters in shifting;
86627 other characters shall not be affected.

86628 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode
86629 buffer.

86630 This command shall be affected by the **autoprint** edit option.

86631 *Current line:* Set to the last line in the lines specified.

86632 *Current column:* Set to non-<blank>.

86633 **Shift Right**

86634 *Synopsis:* **[2addr] >[> ...] [count] [flags]**

86635 Shift the specified lines away from the start of the line; the number of column positions to be
86636 shifted shall be the number of command characters times the value of the **shiftwidth** edit
86637 option. The shift shall be accomplished by adding <blank> characters as a prefix to the line or
86638 changing leading <blank> characters into other <blank> characters. Empty lines shall not be
86639 changed.

86640 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode
86641 buffer.

86642 This command shall be affected by the **autoprint** edit option.

86643 *Current line:* Set to the last line in the lines specified.

86644 *Current column:* Set to non-<blank>.

86645 **<control>-D**

86646 *Synopsis:* **<control>-D**

86647 Write the next *n* lines, where *n* is the minimum of the values of the **scroll** edit option and the
86648 number of lines after the current line in the edit buffer. If the current line is the last line of the
86649 edit buffer it shall be an error.

86650 *Current line:* Set to the last line written.

86651 *Current column:* Set to non-<blank>.

Write Line Number

Synopsis: `[1addr] = [flags]`

If *line* is not specified, it shall default to the last line in the edit buffer. Write the line number of the specified line.

Current line: Unchanged.

Current column: Unchanged.

Execute

Synopsis: `[2addr] @ buffer`
 `[2addr] * buffer`

If no buffer is specified or is specified as `'@'` or `'**'`, the last buffer executed shall be used. If no previous buffer has been executed, it shall be an error.

For each line specified by the addresses, set the current line (`'.'`) to the specified line, and execute the contents of the named *buffer* (as they were at the time the `@` command was executed) as *ex* commands. For each line of a line-mode buffer, and all but the last line of a character-mode buffer, the *ex* command parser shall behave as if the line was terminated by a `<newline>`.

If an error occurs during this process, or a line specified by the addresses does not exist when the current line would be set to it, or more than a single line was specified by the addresses, and the contents of the edit buffer are replaced (for example, by the *ex* **:edit** command) an error message shall be written, and no more commands resulting from the execution of this command shall be processed.

Current line: As specified for the individual *ex* commands.

Current column: As specified for the individual *ex* commands.

Regular Expressions in ex

The *ex* utility shall support regular expressions that are a superset of the basic regular expressions described in XBD [Section 9.3](#) (on page 183). A null regular expression (`"//"`) shall be equivalent to the last regular expression encountered.

Regular expressions can be used in addresses to specify lines and, in some commands (for example, the **substitute** command), to specify portions of a line to be substituted.

The following constructs can be used to enhance the basic regular expressions:

`\<` Match the beginning of a *word*. (See the definition of *word* at the beginning of [Command Descriptions in ex](#) (on page 2651).)

`\>` Match the end of a *word*.

`~` Match the replacement part of the last **substitute** command. The `<tilde>` (`'~'`) character can be escaped in a regular expression to become a normal character with no special meaning. The `<backslash>` shall be discarded.

When the editor option **magic** is not set, the only characters with special meanings shall be `'^'` at the beginning of a pattern, `'$'` at the end of a pattern, and `<backslash>`. The characters `'.'`, `'*'`, `'['`, and `'~'` shall be treated as ordinary characters unless preceded by a `<backslash>`; when preceded by a `<backslash>` they shall regain their special meaning, or in the case of `<backslash>`, be handled as a single `<backslash>`. `<backslash>` characters used to escape other characters shall be discarded.

Replacement Strings in ex

The character '**&**' ('**\&**' if the editor option **magic** is not set) in the replacement string shall stand for the text matched by the pattern to be replaced. The character '**~**' ('**\~**' if **magic** is not set) shall be replaced by the replacement part of the previous **substitute** command. The sequence '**\n**', where *n* is an integer, shall be replaced by the text matched by the corresponding back-reference expression. If the corresponding back-reference expression does not match, then the characters '**\n**' shall be replaced by the empty string.

The strings '**\l**', '**\u**', '**\L**', and '**\U**' can be used to modify the case of elements in the replacement string (using the '**\&**' or "****"digit) notation. The string '**\l**' ('**\u**') shall cause the character that follows to be converted to lowercase (uppercase). The string '**\L**' ('**\U**') shall cause all characters subsequent to it to be converted to lowercase (uppercase) as they are inserted by the substitution until the string '**\e**' or '**\E**', or the end of the replacement string, is encountered.

Otherwise, any character following a <backslash> shall be treated as that literal character, and the escaping <backslash> shall be discarded.

An example of case conversion with the **s** command is as follows:

```
:p
The cat sat on the mat.
:s/\<.at\>/\u&/gp
The Cat Sat on the Mat.
:s/S\ (.*)M/S\U\1\eM/p
The Cat SAT ON THE Mat.
```

Edit Options in ex

The *ex* utility has a number of options that modify its behavior. These options have default settings, which can be changed using the **set** command.

Options are Boolean unless otherwise specified.

autoindent, ai

[Default *unset*]

If **autoindent** is set, each line in input mode shall be indented (using first as many <tab> characters as possible, as determined by the editor option **tabstop**, and then using <space> characters) to align with another line, as follows:

1. If in open or visual mode and the text input is part of a line-oriented command (see the EXTENDED DESCRIPTION in *vi*), align to the first column.
2. Otherwise, if in open or visual mode, indentation for each line shall be set as follows:
 - a. If a line was previously inserted as part of this command, it shall be set to the indentation of the last inserted line by default, or as otherwise specified for the <control>-D character in *Input Mode Commands* in *vi* (on page 3344).
 - b. Otherwise, it shall be set to the indentation of the previous current line, if any; otherwise, to the first column.
3. For the *ex* **a**, **i**, and **c** commands, indentation for each line shall be set as follows:

- 86733 a. If a line was previously inserted as part of this command, it shall be set to the
86734 indentation of the last inserted line by default, or as otherwise specified for the *eof*
86735 character in *Scroll* (on page 2650).
- 86736 b. Otherwise, if the command is the *ex a* command, it shall be set to the line
86737 appended after, if any; otherwise to the first column.
- 86738 c. Otherwise, if the command is the *ex i* command, it shall be set to the line inserted
86739 before, if any; otherwise to the first column.
- 86740 d. Otherwise, if the command is the *ex c* command, it shall be set to the indentation of
86741 the line replaced.

86742 **autoprint, ap**

86743 [Default *set*]

86744 If **autoprint** is set, the current line shall be written after each *ex* command that modifies the
86745 contents of the current edit buffer, and after each **tag** command for which the tag search pattern
86746 was found or tag line number was valid, unless:

- 86747 1. The command was executed while in open or visual mode.
- 86748 2. The command was executed as part of a **global** or **v** command or **@** buffer execution.
- 86749 3. The command was the form of the **read** command that reads a file into the edit buffer.
- 86750 4. The command was the **append**, **change**, or **insert** command.
- 86751 5. The command was not terminated by a **<newline>**.
- 86752 6. The current line shall be written by a flag specified to the command; for example, **delete #**
86753 shall write the current line as specified for the flag modifier to the **delete** command, and
86754 not as specified by the **autoprint** edit option.

86755 **autowrite, aw**

86756 [Default *unset*]

86757 If **autowrite** is set, and the edit buffer has been modified since it was last completely written to
86758 any file, the contents of the edit buffer shall be written as if the *ex write* command had been
86759 specified without arguments, before each command affected by the **autowrite** edit option is
86760 executed. Appending the character **' ! '** to the command name of any of the *ex* commands
86761 except **' ! '** shall prevent the write. If the write fails, it shall be an error and the command shall
86762 not be executed.

86763 **beautify, bf**

86764 XSI [Default *unset*]

86765 If **beautify** is set, all non-printable characters, other than **<tab>**, **<newline>**, and **<form-feed>**
86766 characters, shall be discarded from text read in from files.

- 86767 **directory, dir**
 86768 [Default *implementation-defined*]
 86769 The value of this option specifies the directory in which the editor buffer is to be placed. If this
 86770 directory is not writable by the user, the editor shall quit.
- 86771 **edcompatible, ed**
 86772 [Default *unset*]
 86773 Causes the presence of **g** and **c** suffixes on substitute commands to be remembered, and toggled
 86774 by repeating the suffixes.
- 86775 **errorbells, eb**
 86776 [Default *unset*]
 86777 If the editor is in *ex* mode, and the terminal does not support a standout mode (such as inverse
 86778 video), and **errorbells** is set, error messages shall be preceded by alerting the terminal.
- 86779 **exrc**
 86780 [Default *unset*]
 86781 If **exrc** is set, *ex* shall access any **.exrc** file in the current directory, as described in [Initialization in](#)
 86782 [ex and vi](#) (on page 2642). If **exrc** is not set, *ex* shall ignore any **.exrc** file in the current directory
 86783 during initialization, unless the current directory is that named by the *HOME* environment
 86784 variable.
- 86785 **ignorecase, ic**
 86786 [Default *unset*]
 86787 If **ignorecase** is set, characters that have uppercase and lowercase representations shall have
 86788 those representations considered as equivalent for purposes of regular expression comparison.
 86789 The **ignorecase** edit option shall affect all remembered regular expressions; for example,
 86790 unsetting the **ignorecase** edit option shall cause a subsequent *vi n* command to search for the
 86791 last basic regular expression in a case-sensitive fashion.
- 86792 **list**
 86793 [Default *unset*]
 86794 If **list** is set, edit buffer lines written while in *ex* command mode shall be written as specified for
 86795 the **print** command with the **l** flag specified. In open or visual mode, each edit buffer line shall
 86796 be displayed as specified for the *ex print* command with the **l** flag specified. In open or visual
 86797 text input mode, when the cursor does not rest on any character in the line, it shall rest on the
 86798 ' \$ ' marking the end of the line.

86799	magic
86800	[Default <i>set</i>]
86801	If magic is set, modify the interpretation of characters in regular expressions and substitution
86802	replacement strings (see Regular Expressions in ex (on page 2675) and Replacement Strings in
86803	ex , on page 2676).
86804	mesg
86805	[Default <i>set</i>]
86806	If mesg is set, the permission for others to use the write or talk commands to write to the
86807	terminal shall be turned on while in open or visual mode. The shell-level command <i>mesg n</i> shall
86808	take precedence over any setting of the <i>ex mesg</i> option; that is, if mesg y was issued before the
86809	editor started (or in a shell escape), such as:
86810	:!mesg y
86811	the mesg option in <i>ex</i> shall suppress incoming messages, but the mesg option shall not enable
86812	incoming messages if mesg n was issued.
86813	number, nu
86814	[Default <i>unset</i>]
86815	If number is set, edit buffer lines written while in <i>ex</i> command mode shall be written with line
86816	numbers, in the format specified by the print command with the # flag specified. In <i>ex</i> text input
86817	mode, each line shall be preceded by the line number it will have in the file.
86818	In open or visual mode, each edit buffer line shall be displayed with a preceding line number, in
86819	the format specified by the <i>ex print</i> command with the # flag specified. This line number shall
86820	not be considered part of the line for the purposes of evaluating the current column; that is,
86821	column position 1 shall be the first column position after the format specified by the print
86822	command.
86823	paragraphs, para
86824	[Default in the POSIX locale IPLPPPQPP LIpplpipbp]
86825	The paragraphs edit option shall define additional paragraph boundaries for the open and
86826	visual mode commands. The paragraphs edit option can be set to a character string consisting of
86827	zero or more character pairs. It shall be an error to set it to an odd number of characters.
86828	prompt
86829	[Default <i>set</i>]
86830	If prompt is set, <i>ex</i> command mode input shall be prompted for with a <colon> (':'); when
86831	unset, no prompt shall be written.

readonly[Default *see text*]

If the **readonly** edit option is set, read-only mode shall be enabled (see [Write](#), on page 2670). The **readonly** edit option shall be initialized to set if either of the following conditions are true:

- The command-line option **-R** was specified.
- Performing actions equivalent to the *access()* function called with the following arguments indicates that the file lacks write permission:
 1. The current pathname is used as the *path* argument.
 2. The constant **W_OK** is used as the *amode* argument.

The **readonly** edit option may be initialized to set for other, implementation-defined reasons. The **readonly** edit option shall not be initialized to unset based on any special privileges of the user or process. The **readonly** edit option shall be reinitialized each time that the contents of the edit buffer are replaced (for example, by an **edit** or **next** command) unless the user has explicitly set it, in which case it shall remain set until the user explicitly unsets it. Once unset, it shall again be reinitialized each time that the contents of the edit buffer are replaced.

redraw[Default *unset*]

The editor simulates an intelligent terminal on a dumb terminal. (Since this is likely to require a large amount of output to the terminal, it is useful only at high transmission speeds.)

remap[Default *set*]

If **remap** is set, map translation shall allow for maps defined in terms of other maps; translation shall continue until a final product is obtained. If unset, only a one-step translation shall be done.

report

[Default 5]

The value of this **report** edit option specifies what number of lines being added, copied, deleted, or modified in the edit buffer will cause an informational message to be written to the user. The following conditions shall cause an informational message. The message shall contain the number of lines added, copied, deleted, or modified, but is otherwise unspecified.

- An *ex* or *vi* editor command, other than **open**, **undo**, or **visual**, that modifies at least the value of the **report** edit option number of lines, and which is not part of an *ex* **global** or **v** command, or *ex* or *vi* buffer execution, shall cause an informational message to be written.
- An *ex* **yank** or *vi* **y** or **Y** command, that copies at least the value of the **report** edit option plus 1 number of lines, and which is not part of an *ex* **global** or **v** command, or *ex* or *vi* buffer execution, shall cause an informational message to be written.
- An *ex* **global**, **v**, **open**, **undo**, or **visual** command or *ex* or *vi* buffer execution, that adds or deletes a total of at least the value of the **report** edit option number of lines, and which is not part of an *ex* **global** or **v** command, or *ex* or *vi* buffer execution, shall cause an informational message to be written. (For example, if 3 lines were added and 8 lines deleted during an *ex* **visual** command, 5 would be the number compared against the

report edit option after the command completed.)

scroll, scr

[Default (number of lines in the display -1)/2]

The value of the **scroll** edit option shall determine the number of lines scrolled by the *ex* <control>-D and *z* commands. For the *vi* <control>-D and <control>-U commands, it shall be the initial number of lines to scroll when no previous <control>-D or <control>-U command has been executed.

sections

[Default in the POSIX locale `NHSHH HUnhsh`]

The **sections** edit option shall define additional section boundaries for the open and visual mode commands. The **sections** edit option can be set to a character string consisting of zero or more character pairs; it shall be an error to set it to an odd number of characters.

shell, sh

[Default from the environment variable *SHELL*]

The value of this option shall be a string. The default shall be taken from the *SHELL* environment variable. If the *SHELL* environment variable is null or empty, the *sh* (see *sh*) utility shall be the default.

shiftwidth, sw

[Default 8]

The value of this option shall give the width in columns of an indentation level used during autoindentation and by the shift commands (< and >).

showmatch, sm

[Default *unset*]

The functionality described for the **showmatch** edit option need not be supported on block-mode terminals or terminals with insufficient capabilities.

If **showmatch** is set, in open or visual mode, when a ') ' or ' } ' is typed, if the matching ' (' or ' { ' is currently visible on the display, the matching ' (' or ' { ' shall be flagged moving the cursor to its location for an unspecified amount of time.

showmode

[Default *unset*]

If **showmode** is set, in open or visual mode, the current mode that the editor is in shall be displayed on the last line of the display. Command mode and text input mode shall be differentiated; other unspecified modes and implementation-defined information may be displayed.

- 86907 **slowopen**
- 86908 [Default *unset*]
- 86909 If **slowopen** is set during open and visual text input modes, the editor shall not update portions
86910 of the display other than those display line columns that display the characters entered by the
86911 user (see [Input Mode Commands in vi](#), on page 3344).
- 86912 **tabstop, ts**
- 86913 [Default 8]
- 86914 The value of this edit option shall specify the column boundary used by a <tab> in the display
86915 (see [autoprint, ap](#) (on page 2677) and [Input Mode Commands in vi](#), on page 3344).
- 86916 **taglength, tl**
- 86917 [Default zero]
- 86918 The value of this edit option shall specify the maximum number of characters that are
86919 considered significant in the user-specified tag name and in the tag name from the tags file. If
86920 the value is zero, all characters in both tag names shall be significant.
- 86921 **tags**
- 86922 [Default *see text*]
- 86923 The value of this edit option shall be a string of <blank>-delimited pathnames of files used by
86924 the **tag** command. The default value is unspecified.
- 86925 **term**
- 86926 [Default from the environment variable *TERM*]
- 86927 The value of this edit option shall be a string. The default shall be taken from the *TERM* variable
86928 in the environment. If the *TERM* environment variable is empty or null, the default is
86929 unspecified. The editor shall use the value of this edit option to determine the type of the display
86930 device.
- 86931 The results are unspecified if the user changes the value of the term edit option after editor
86932 initialization.
- 86933 **terse**
- 86934 [Default *unset*]
- 86935 If **terse** is set, error messages may be less verbose. However, except for this caveat, error
86936 messages are unspecified. Furthermore, not all error messages need change for different settings
86937 of this option.

warn[Default *set*]

If **warn** is set, and the contents of the edit buffer have been modified since they were last completely written, the editor shall write a warning message before certain ! commands (see [Escape](#), on page 2673).

window[Default *see text*]

A value used in open and visual mode, by the <control>-B and <control>-F commands, and, in visual mode, to specify the number of lines displayed when the screen is repainted.

If the **-w** command-line option is not specified, the default value shall be set to the value of the *LINES* environment variable. If the *LINES* environment variable is empty or null, the default shall be the number of lines in the display minus 1.

Setting the **window** edit option to zero or to a value greater than the number of lines in the display minus 1 (either explicitly or based on the **-w** option or the *LINES* environment variable) shall cause the **window** edit option to be set to the number of lines in the display minus 1.

The baud rate of the terminal line may change the default in an implementation-defined manner.

wrapmargin, wm

[Default 0]

If the value of this edit option is zero, it shall have no effect.

If not in the POSIX locale, the effect of this edit option is implementation-defined.

Otherwise, it shall specify a number of columns from the ending margin of the terminal.

During open and visual text input modes, for each character for which any part of the character is displayed in a column that is less than **wrapmargin** columns from the ending margin of the display line, the editor shall behave as follows:

1. If the character triggering this event is a <blank>, it, and all immediately preceding <blank> characters on the current line entered during the execution of the current text input command, shall be discarded, and the editor shall behave as if the user had entered a single <newline> instead. In addition, if the next user-entered character is a <space>, it shall be discarded as well.
2. Otherwise, if there are one or more <blank> characters on the current line immediately preceding the last group of inserted non-<blank> characters which was entered during the execution of the current text input command, the <blank> characters shall be replaced as if the user had entered a single <newline> instead.

If the **autoindent** edit option is set, and the events described in 1. or 2. are performed, any <blank> characters at or after the cursor in the current line shall be discarded.

The ending margin shall be determined by the system or overridden by the user, as described for *COLUMNS* in the ENVIRONMENT VARIABLES section and XBD [Chapter 8](#) (on page 173).

86975 **wrapsan, ws**

86976 [Default *set*]

86977 If **wrapsan** is set, searches (the *ex* / or ? addresses, or open and visual mode /, ?, N, and n
86978 commands) shall wrap around the beginning or end of the edit buffer; when unset, searches
86979 shall stop at the beginning or end of the edit buffer.

86980 **writeany, wa**

86981 [Default *unset*]

86982 If **writeany** is set, some of the checks performed when executing the *ex* **write** commands shall be
86983 inhibited, as described in editor option **autowrite**.

86984 EXIT STATUS

86985 The following exit values shall be returned:

86986 0 Successful completion.

86987 >0 An error occurred.

86988 CONSEQUENCES OF ERRORS

86989 When any error is encountered and the standard input is not a terminal device file, *ex* shall not
86990 write the file or return to command or text input mode, and shall terminate with a non-zero exit
86991 status.

86992 Otherwise, when an unrecoverable error is encountered, it shall be equivalent to a SIGHUP
86993 asynchronous event.

86994 Otherwise, when an error is encountered, the editor shall behave as specified in [Command Line](#)
86995 [Parsing in ex](#) (on page 2646).

86996 APPLICATION USAGE

86997 If a SIGSEGV signal is received while *ex* is saving a file, the file might not be successfully saved.

86998 The **next** command can accept more than one file, so usage such as:

86999 `next `ls [abc]*``

87000 is valid; it would not be valid for the **edit** or **read** commands, for example, because they expect
87001 only one file and unspecified results occur.

87002 EXAMPLES

87003 None.

87004 RATIONALE

87005 The *ex/vi* specification is based on the historical practice found in the 4 BSD and System V
87006 implementations of *ex* and *vi*.

87007 A *restricted editor* (both the historical *red* utility and modifications to *ex*) were considered and
87008 rejected for inclusion. Neither option provided the level of security that users might expect.

87009 It is recognized that *ex* visual mode and related features would be difficult, if not impossible, to
87010 implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor
87011 addressing; thus, it is not a mandatory requirement that such features should work on all
87012 terminals. It is the intention, however, that an *ex* implementation should provide the full set of
87013 capabilities on all terminals capable of supporting them.

Options

The `-c` replacement for `+command` was inspired by the `-e` option of `sed`. Historically, all such commands (see `edit` and `next` as well) were executed from the last line of the edit buffer. This meant, for example, that `"/pattern"` would fail unless the `wrapscan` option was set. POSIX.1-2008 requires conformance to historical practice. The `+command` option is no longer specified by POSIX.1-2008 but may be present in some implementations. Historically, some implementations restricted the `ex` commands that could be listed as part of the command line arguments. For consistency, POSIX.1-2008 does not permit these restrictions.

In historical implementations of the editor, the `-R` option (and the `readonly` edit option) only prevented overwriting of files; appending to files was still permitted, mapping loosely into the `cs` `noclobber` variable. Some implementations, however, have not followed this semantic, and `readonly` does not permit appending either. POSIX.1-2008 follows the latter practice, believing that it is a more obvious and intuitive meaning of `readonly`.

The `-s` option suppresses all interactive user feedback and is useful for editing scripts in batch jobs. The list of specific effects is historical practice. The terminal type "incapable of supporting open and visual modes" has historically been named "dumb".

The `-t` option was required because the `ctags` utility appears in POSIX.1-2008 and the option is available in all historical implementations of `ex`.

Historically, the `ex` and `vi` utilities accepted a `-x` option, which did encryption based on the algorithm found in the historical `crypt` utility. The `-x` option for encryption, and the associated `crypt` utility, were omitted because the algorithm used was not specifiable and the export control laws of some nations make it difficult to export cryptographic technology. In addition, it did not historically provide the level of security that users might expect.

Standard Input

An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file character, `<control>-D`, is historically an `ex` command.

There was no maximum line length in historical implementations of `ex`. Specifically, as it was parsed in chunks, the addresses had a different maximum length than the filenames. Further, the maximum line buffer size was declared as `BUFSIZ`, which was different lengths on different systems. This version selected the value of `{LINE_MAX}` to impose a reasonable restriction on portable usage of `ex` and to aid test suite writers in their development of realistic tests that exercise this limit.

Input Files

It was an explicit decision by the standard developers that a `<newline>` be added to any file lacking one. It was believed that this feature of `ex` and `vi` was relied on by users in order to make text files lacking a trailing `<newline>` more portable. It is recognized that this will require a user-specified option or extension for implementations that permit `ex` and `vi` to edit files of type other than text if such files are not otherwise identified by the system. It was agreed that the ability to edit files of arbitrary type can be useful, but it was not considered necessary to mandate that an `ex` or `vi` implementation be required to handle files other than text files.

The paragraph in the INPUT FILES section, "By default, ...", is intended to close a long-standing security problem in `ex` and `vi`; that of the "modeline" or "modelines" edit option. This feature allows any line in the first or last five lines of the file containing the strings `"ex:"` or `"vi:"` (and, apparently, `"ei:"` or `"vx:"`) to be a line containing editor commands, and `ex` interprets all the text up to the next `':'` or `<newline>` as a command. Consider the

consequences, for example, of an unsuspecting user using *ex* or *vi* as the editor when replying to a mail message in which a line such as:

```
ex:! rm -rf :
```

appeared in the signature lines. The standard developers believed strongly that an editor should not by default interpret any lines of a file. Vendors are strongly urged to delete this feature from their implementations of *ex* and *vi*.

Asynchronous Events

The intention of the phrase “complete write” is that the entire edit buffer be written to stable storage. The note regarding temporary files is intended for implementations that use temporary files to back edit buffers unnamed by the user.

Historically, SIGQUIT was ignored by *ex*, but was the equivalent of the Q command in visual mode; that is, it exited visual mode and entered *ex* mode. POSIX.1-2008 permits, but does not require, this behavior. Historically, SIGINT was often used by *vi* users to terminate text input mode (<control>-C is often easier to enter than <ESC>). Some implementations of *vi* alerted the terminal on this event, and some did not. POSIX.1-2008 requires that SIGINT behave identically to <ESC>, and that the terminal not be alerted.

Historically, suspending the *ex* editor during text input mode was similar to SIGINT, as completed lines were retained, but any partial line discarded, and the editor returned to command mode. POSIX.1-2008 is silent on this issue; implementations are encouraged to follow historical practice, where possible.

Historically, the *vi* editor did not treat SIGTSTP as an asynchronous event, and it was therefore impossible to suspend the editor in visual text input mode. There are two major reasons for this. The first is that SIGTSTP is a broadcast signal on UNIX systems, and the chain of events where the shell *execs* an application that then *execs vi* usually caused confusion for the terminal state if SIGTSTP was delivered to the process group in the default manner. The second was that most implementations of the UNIX *curses* package did not handle SIGTSTP safely, and the receipt of SIGTSTP at the wrong time would cause them to crash. POSIX.1-2008 is silent on this issue; implementations are encouraged to treat suspension as an asynchronous event if possible.

Historically, modifications to the edit buffer made before SIGINT interrupted an operation were retained; that is, anywhere from zero to all of the lines to be modified might have been modified by the time the SIGINT arrived. These changes were not discarded by the arrival of SIGINT. POSIX.1-2008 permits this behavior, noting that the **undo** command is required to be able to undo these partially completed commands.

The action taken for signals other than SIGINT, SIGCONT, SIGHUP, and SIGTERM is unspecified because some implementations attempt to save the edit buffer in a useful state when other signals are received.

Standard Error

For *ex/vi*, diagnostic messages are those messages reported as a result of a failed attempt to invoke *ex* or *vi*, such as invalid options or insufficient resources, or an abnormal termination condition. Diagnostic messages should not be confused with the error messages generated by inappropriate or illegal user commands.

Initialization in ex and vi

If an *ex* command (other than **cd**, **chdir**, or **source**) has a filename argument, one or both of the alternate and current pathnames will be set. Informally, they are set as follows:

1. If the *ex* command is one that replaces the contents of the edit buffer, and it succeeds, the current pathname will be set to the filename argument (the first filename argument in the case of the **next** command) and the alternate pathname will be set to the previous current pathname, if there was one.
2. In the case of the file read/write forms of the **read** and **write** commands, if there is no current pathname, the current pathname will be set to the filename argument.
3. Otherwise, the alternate pathname will be set to the filename argument.

For example, **:edit foo** and **:recover foo**, when successful, set the current pathname, and, if there was a previous current pathname, the alternate pathname. The commands **:write**, **!command**, and **:edit** set neither the current or alternate pathnames. If the **:edit foo** command were to fail for some reason, the alternate pathname would be set. The **read** and **write** commands set the alternate pathname to their *file* argument, unless the current pathname is not set, in which case they set the current pathname to their *file* arguments. The alternate pathname was not historically set by the **:source** command. POSIX.1-2008 requires conformance to historical practice. Implementations adding commands that take filenames as arguments are encouraged to set the alternate pathname as described here.

Historically, *ex* and *vi* read the **.exrc** file in the *\$HOME* directory twice, if the editor was executed in the *\$HOME* directory. POSIX.1-2008 prohibits this behavior.

Historically, the 4 BSD *ex* and *vi* read the *\$HOME* and local **.exrc** files if they were owned by the real ID of the user, or the **sourceany** option was set, regardless of other considerations. This was a security problem because it is possible to put normal UNIX system commands inside a **.exrc** file. POSIX.1-2008 does not specify the **sourceany** option, and historical implementations are encouraged to delete it.

The **.exrc** files must be owned by the real ID of the user, and not writable by anyone other than the owner. The appropriate privileges exception is intended to permit users to acquire special privileges, but continue to use the **.exrc** files in their home directories.

System V Release 3.2 and later *vi* implementations added the option **[no]exrc**. The behavior is that local **.exrc** files are read-only if the **exrc** option is set. The default for the **exrc** option was off, so by default, local **.exrc** files were not read. The problem this was intended to solve was that System V permitted users to give away files, so there is no possible ownership or writeability test to ensure that the file is safe. This is still a security problem on systems where users can give away files, but there is nothing additional that POSIX.1-2008 can do. The implementation-defined exception is intended to permit groups to have local **.exrc** files that are shared by users, by creating pseudo-users to own the shared files.

POSIX.1-2008 does not mention system-wide *ex* and *vi* start-up files. While they exist in several implementations of *ex* and *vi*, they are not present in any implementations considered historical practice by POSIX.1-2008. Implementations that have such files should use them only if they are owned by the real user ID or an appropriate user (for example, root on UNIX systems) and if they are not writable by any user other than their owner. System-wide start-up files should be read before the *EXINIT* variable, *\$HOME/.exrc*, or local **.exrc** files are evaluated.

Historically, any *ex* command could be entered in the *EXINIT* variable or the **.exrc** file, although ones requiring that the edit buffer already contain lines of text generally caused historical implementations of the editor to drop **core**. POSIX.1-2008 requires that any *ex* command be

permitted in the *EXINIT* variable and *.exrc* files, for simplicity of specification and consistency, although many of them will obviously fail under many circumstances.

The initialization of the contents of the edit buffer uses the phrase “the effect shall be” with regard to various *ex* commands. The intent of this phrase is that edit buffer contents loaded during the initialization phase not be lost; that is, loading the edit buffer should fail if the *.exrc* file read in the contents of a file and did not subsequently write the edit buffer. An additional intent of this phrase is to specify that the initial current line and column is set as specified for the individual *ex* commands.

Historically, the *-t* option behaved as if the tag search were a *+command*; that is, it was executed from the last line of the file specified by the tag. This resulted in the search failing if the pattern was a forward search pattern and the *wraps*can edit option was not set. POSIX.1-2008 does not permit this behavior, requiring that the search for the tag pattern be performed on the entire file, and, if not found, that the current line be set to a more reasonable location in the file.

Historically, the empty edit buffer presented for editing when a file was not specified by the user was unnamed. This is permitted by POSIX.1-2008; however, implementations are encouraged to provide users a temporary filename for this buffer because it permits them the use of *ex* commands that use the current pathname during temporary edit sessions.

Historically, the file specified using the *-t* option was not part of the current argument list. This practice is permitted by POSIX.1-2008; however, implementations are encouraged to include its name in the current argument list for consistency.

Historically, the *-c* command was generally not executed until a file that already exists was edited. POSIX.1-2008 requires conformance to this historical practice. Commands that could cause the *-c* command to be executed include the *ex* commands *edit*, *next*, *recover*, *rewind*, and *tag*, and the *vi* commands *<control>-^* and *<control>-]*. Historically, reading a file into an edit buffer did not cause the *-c* command to be executed (even though it might set the current pathname) with the exception that it did cause the *-c* command to be executed if: the editor was in *ex* mode, the edit buffer had no current pathname, the edit buffer was empty, and no read commands had yet been attempted. For consistency and simplicity of specification, POSIX.1-2008 does not permit this behavior.

Historically, the *-r* option was the same as a normal edit session if there was no recovery information available for the file. This allowed users to enter:

```
vi -r *.c
```

and recover whatever files were recoverable. In some implementations, recovery was attempted only on the first file named, and the file was not entered into the argument list; in others, recovery was attempted for each file named. In addition, some historical implementations ignored *-r* if *-t* was specified or did not support command line *file* arguments with the *-t* option. For consistency and simplicity of specification, POSIX.1-2008 disallows these special cases, and requires that recovery be attempted the first time each file is edited.

Historically, *vi* initialized the ‘ and ’ marks, but *ex* did not. This meant that if the first command in *ex* mode was *visual* or if an *ex* command was executed first (for example, *vi +10 file*), *vi* was entered without the marks being initialized. Because the standard developers believed the marks to be generally useful, and for consistency and simplicity of specification, POSIX.1-2008 requires that they always be initialized if in open or visual mode, or if in *ex* mode and the edit buffer is not empty. Not initializing it in *ex* mode if the edit buffer is empty is historical practice; however, it has always been possible to set (and use) marks in empty edit buffers in open and visual mode edit sessions.

Addressing

Historically, *ex* and *vi* accepted the additional addressing forms '`\`' and '`\?`'. They were equivalent to '`//`' and '`??`', respectively. They are not required by POSIX.1-2008, mostly because nobody can remember whether they ever did anything different historically.

Historically, *ex* and *vi* permitted an address of zero for several commands, and permitted the % address in empty files for others. For consistency, POSIX.1-2008 requires support for the former in the few commands where it makes sense, and disallows it otherwise. In addition, because POSIX.1-2008 requires that % be logically equivalent to "1, \$", it is also supported where it makes sense and disallowed otherwise.

Historically, the % address could not be followed by further addresses. For consistency and simplicity of specification, POSIX.1-2008 requires that additional addresses be supported.

All of the following are valid *addresses*:

+++	Three lines after the current line.
/re/-	One line before the next occurrence of <i>re</i> .
-2	Two lines before the current line.
3 ---- 2	Line one (note intermediate negative address).
1 2 3	Line six.

Any number of addresses can be provided to commands taking addresses; for example, "`1, 2, 3, 4, 5p`" prints lines 4 and 5, because two is the greatest valid number of addresses accepted by the **print** command. This, in combination with the <semicolon> delimiter, permits users to create commands based on ordered patterns in the file. For example, the command **3;/foo/;+2print** will display the first line after line 3 that contains the pattern *foo*, plus the next two lines. Note that the address **3;** must be evaluated before being discarded because the search origin for the **/foo/** command depends on this.

Historically, values could be added to addresses by including them after one or more <blank> characters; for example, `3 -5p` wrote the seventh line of the file, and **/foo/ 5** was the same as **/foo/+5**. However, only absolute values could be added; for example, `5 /foo/` was an error. POSIX.1-2008 requires conformance to historical practice. Address offsets are separately specified from addresses because they could historically be provided to visual mode search commands.

Historically, any missing addresses defaulted to the current line. This was true for leading and trailing <comma>-delimited addresses, and for trailing <semicolon>-delimited addresses. For consistency, POSIX.1-2008 requires it for leading <semicolon> addresses as well.

Historically, *ex* and *vi* accepted the '`^`' character as both an address and as a flag offset for commands. In both cases it was identical to the '`-`' character. POSIX.1-2008 does not require or prohibit this behavior.

Historically, the enhancements to basic regular expressions could be used in addressing; for example, '`~`', '`\<`', and '`\>`'. POSIX.1-2008 requires conformance to historical practice; that is, that regular expression usage be consistent, and that regular expression enhancements be supported wherever regular expressions are used.

Command Line Parsing in ex

Historical *ex* command parsing was even more complex than that described here. POSIX.1-2008 requires the subset of the command parsing that the standard developers believed was documented and that users could reasonably be expected to use in a portable fashion, and that was historically consistent between implementations. (The discarded functionality is obscure, at best.) Historical implementations will require changes in order to comply with POSIX.1-2008; however, users are not expected to notice any of these changes. Most of the complexity in *ex* parsing is to handle three special termination cases:

1. The **!**, **global**, **v**, and the filter versions of the **read** and **write** commands are delimited by <newline> characters (they can contain <vertical-line> characters that are usually shell pipes).
2. The **ex**, **edit**, **next**, and **visual** in open and visual mode commands all take *ex* commands, optionally containing <vertical-line> characters, as their first arguments.
3. The **s** command takes a regular expression as its first argument, and uses the delimiting characters to delimit the command.

Historically, <vertical-line> characters in the *+command* argument of the **ex**, **edit**, **next**, **vi**, and **visual** commands, and in the *pattern* and *replacement* parts of the **s** command, did not delimit the command, and in the filter cases for **read** and **write**, and the **!**, **global**, and **v** commands, they did not delimit the command at all. For example, the following commands are all valid:

```
:edit +25 | s/abc/ABC/ file.c
:s/ | /PIPE/
:read !spell % | columnate
:global/pattern/p | l
:s/a/b/ | s/c/d | set
```

Historically, empty or <blank> filled lines in **.exrc** files and **sourced** files (as well as *EXINIT* variables and *ex* command scripts) were treated as default commands; that is, **print** commands. POSIX.1-2008 specifically requires that they be ignored when encountered in **.exrc** and **sourced** files to eliminate a common source of new user error.

Historically, *ex* commands with multiple adjacent (or <blank>-separated) vertical lines were handled oddly when executed from *ex* mode. For example, the command **l l l** <carriage-return>, when the cursor was on line 1, displayed lines 2, 3, and 5 of the file. In addition, the command **l** would only display the line after the next line, instead of the next two lines. The former worked more logically when executed from *vi* mode, and displayed lines 2, 3, and 4. POSIX.1-2008 requires the *vi* behavior; that is, a single default command and line number increment for each command separator, and trailing <newline> characters after <vertical-line> separators are discarded.

Historically, *ex* permitted a single extra <colon> as a leading command character; for example, **:g/pattern/:p** was a valid command. POSIX.1-2008 generalizes this to require that any number of leading <colon> characters be stripped.

Historically, any prefix of the **delete** command could be followed without intervening <blank> characters by a flag character because in the command **d p**, *p* is interpreted as the buffer *p*. POSIX.1-2008 requires conformance to historical practice.

Historically, the **k** command could be followed by the mark name without intervening <blank> characters. POSIX.1-2008 requires conformance to historical practice.

Historically, the **s** command could be immediately followed by flag and option characters; for example, **s/e/E/l s lsgc3p** was a valid command. However, flag characters could not stand alone;

for example, the commands **sp** and **s l** would fail, while the command **sgp** and **s gl** would succeed. (Obviously, the '#' flag character was used as a delimiter character if it followed the command.) Another issue was that option characters had to precede flag characters even when the command was fully specified; for example, the command **s/e/E/pg** would fail, while the command **s/e/E/gp** would succeed. POSIX.1-2008 requires conformance to historical practice.

Historically, the first command name that had a prefix matching the input from the user was the executed command; for example, **ve**, **ver**, and **vers** all executed the **version** command. Commands were in a specific order, however, so that **a** matched **append**, not **abbreviate**. POSIX.1-2008 requires conformance to historical practice. The restriction on command search order for implementations with extensions is to avoid the addition of commands such that the historical prefixes would fail to work portably.

Historical implementations of *ex* and *vi* did not correctly handle multiple *ex* commands, separated by <vertical-line> characters, that entered or exited visual mode of the editor. Because implementations of *vi* exist that do not exhibit this failure mode, POSIX.1-2008 does not permit it.

The requirement that alphabetic command names consist of all following alphabetic characters up to the next non-alphabetic character means that alphabetic command names must be separated from their arguments by one or more non-alphabetic characters, normally a <blank> or '!' character, except as specified for the exceptions, the **delete**, **k**, and **s** commands.

Historically, the repeated execution of the *ex* default **print** commands (<control>-D, *eof*, <newline>, <carriage-return>) erased any prompting character and displayed the next lines without scrolling the terminal; that is, immediately below any previously displayed lines. This provided a cleaner presentation of the lines in the file for the user. POSIX.1-2008 does not require this behavior because it may be impossible in some situations; however, implementations are strongly encouraged to provide this semantic if possible.

Historically, it was possible to change files in the middle of a command, and have the rest of the command executed in the new file, for example:

```
:edit +25 file.c | s/abc/ABC/ | 1
```

was a valid command, and the substitution was attempted in the newly edited file. POSIX.1-2008 requires conformance to historical practice. The following commands are examples that exercise the *ex* parser:

```
echo 'foo | bar' > file1; echo 'foo/bar' > file2;
```

```
vi
```

```
:edit +1 | s/||PIPE/ | w file1 | e file2 | 1 | s/\\//SLASH/ | wq
```

Historically, there was no protection in editor implementations to avoid *ex* **global**, **v**, **@**, or ***** commands changing edit buffers during execution of their associated commands. Because this would almost invariably result in catastrophic failure of the editor, and implementations exist that do exhibit these problems, POSIX.1-2008 requires that changing the edit buffer during a **global** or **v** command, or during a **@** or ***** command for which there will be more than a single execution, be an error. Implementations supporting multiple edit buffers simultaneously are strongly encouraged to apply the same semantics to switching between buffers as well.

The *ex* command quoting required by POSIX.1-2008 is a superset of the quoting in historical implementations of the editor. For example, it was not historically possible to escape a <blank> in a filename; for example, **:edit foo\\ bar** would report that too many filenames had been entered for the edit command, and there was no method of escaping a <blank> in the first argument of an **edit**, **ex**, **next**, or **visual** command at all. POSIX.1-2008 extends historical practice, requiring that quoting behavior be made consistent across all *ex* commands, except for

the **map**, **unmap**, **abbreviate**, and **unabbreviate** commands, which historically used <control>-V instead of <backslash> characters for quoting. For those four commands, POSIX.1-2008 requires conformance to historical practice.

Backslash quoting in *ex* is non-intuitive. <backslash>-escapes are ignored unless they escape a special character; for example, when performing *file* argument expansion, the string "\\%" is equivalent to '\%', not "\<current pathname>". This can be confusing for users because <backslash> is usually one of the characters that causes shell expansion to be performed, and therefore shell quoting rules must be taken into consideration. Generally, quoting characters are only considered if they escape a special character, and a quoting character must be provided for each layer of parsing for which the character is special. As another example, only a single <backslash> is necessary for the '\1' sequence in substitute replacement patterns, because the character '1' is not special to any parsing layer above it.

<control>-V quoting in *ex* is slightly different from backslash quoting. In the four commands where <control>-V quoting applies (**abbreviate**, **unabbreviate**, **map**, and **unmap**), any character may be escaped by a <control>-V whether it would have a special meaning or not. POSIX.1-2008 requires conformance to historical practice.

Historical implementations of the editor did not require delimiters within character classes to be escaped; for example, the command **s[/]/** on the string "xxx/yyy" would delete the '/' from the string. POSIX.1-2008 disallows this historical practice for consistency and because it places a large burden on implementations by requiring that knowledge of regular expressions be built into the editor parser.

Historically, quoting <newline> characters in *ex* commands was handled inconsistently. In most cases, the <newline> character always terminated the command, regardless of any preceding escape character, because <backslash> characters did not escape <newline> characters for most *ex* commands. However, some *ex* commands (for example, **s**, **map**, and **abbreviation**) permitted <newline> characters to be escaped (although in the case of **map** and **abbreviation**, <control>-V characters escaped them instead of <backslash> characters). This was true in not only the command line, but also **.exrc** and **sourced** files. For example, the command:

```
map = foo<control-V><newline>bar
```

would succeed, although it was sometimes difficult to get the <control>-V and the inserted <newline> passed to the *ex* parser. For consistency and simplicity of specification, POSIX.1-2008 requires that it be possible to escape <newline> characters in *ex* commands at all times, using <backslash> characters for most *ex* commands, and using <control>-V characters for the **map** and **abbreviation** commands. For example, the command **print<newline>list** is required to be parsed as the single command **print<newline>list**. While this differs from historical practice, POSIX.1-2008 developers believed it unlikely that any script or user depended on the historical behavior.

Historically, an error in a command specified using the **-c** option did not cause the rest of the **-c** commands to be discarded. POSIX.1-2008 disallows this for consistency with mapped keys, the **@**, **global**, **source**, and **v** commands, the *EXINIT* environment variable, and the **.exrc** files.

Input Editing in *ex*

One of the common uses of the historical *ex* editor is over slow network connections. Editors that run in canonical mode can require far less traffic to and from, and far less processing on, the host machine, as well as more easily supporting block-mode terminals. For these reasons, POSIX.1-2008 requires that *ex* be implemented using canonical mode input processing, as was done historically.

POSIX.1-2008 does not require the historical 4 BSD input editing characters “word erase” or “literal next”. For this reason, it is unspecified how they are handled by *ex*, although they must have the required effect. Implementations that resolve them after the line has been ended using a <newline> or <control>-M character, and implementations that rely on the underlying system terminal support for this processing, are both conforming. Implementations are strongly urged to use the underlying system functionality, if at all possible, for compatibility with other system text input interfaces.

Historically, when the *eof* character was used to decrement the **autoindent** level, the cursor moved to display the new end of the **autoindent** characters, but did not move the cursor to a new line, nor did it erase the <control>-D character from the line. POSIX.1-2008 does not specify that the cursor remain on the same line or that the rest of the line is erased; however, implementations are strongly encouraged to provide the best possible user interface; that is, the cursor should remain on the same line, and any <control>-D character on the line should be erased.

POSIX.1-2008 does not require the historical 4 BSD input editing character “reprint”, traditionally <control>-R, which redisplayed the current input from the user. For this reason, and because the functionality cannot be implemented after the line has been terminated by the user, POSIX.1-2008 makes no requirements about this functionality. Implementations are strongly urged to make this historical functionality available, if possible.

Historically, <control>-Q did not perform a literal next function in *ex*, as it did in *vi*. POSIX.1-2008 requires conformance to historical practice to avoid breaking historical *ex* scripts and *.exrc* files.

eof

Whether the *eof* character immediately modifies the **autoindent** characters in the prompt is left unspecified so that implementations can conform in the presence of systems that do not support this functionality. Implementations are encouraged to modify the line and redisplay it immediately, if possible.

The specification of the handling of the *eof* character differs from historical practice only in that *eof* characters are not discarded if they follow normal characters in the text input. Historically, they were always discarded.

Command Descriptions in *ex*

Historically, several commands (for example, **global**, **v**, **visual**, **s**, **write**, **wq**, **yank**, **!**, **<**, **>**, **&**, and **~**) were executable in empty files (that is, the default address(es) were 0), or permitted explicit addresses of 0 (for example, 0 was a valid address, or 0,0 was a valid range). Addresses of 0, or command execution in an empty file, make sense only for commands that add new text to the edit buffer or write commands (because users may wish to write empty files). POSIX.1-2008 requires this behavior for such commands and disallows it otherwise, for consistency and simplicity of specification.

A count to an *ex* command has been historically corrected to be no greater than the last line in a

file; for example, in a five-line file, the command **1,6print** would fail, but the command **1print300** would succeed. POSIX.1-2008 requires conformance to historical practice.

Historically, the use of flags in *ex* commands could be obscure. General historical practice was as described by POSIX.1-2008, but there were some special cases. For instance, the **list**, **number**, and **print** commands ignored trailing address offsets; for example, **3p +++#** would display line 3, and 3 would be the current line after the execution of the command. The **open** and **visual** commands ignored both the trailing offsets and the trailing flags. Also, flags specified to the **open** and **visual** commands interacted badly with the **list** edit option, and setting and then unsetting it during the open/visual session would cause *vi* to stop displaying lines in the specified format. For consistency and simplicity of specification, POSIX.1-2008 does not permit any of these exceptions to the general rule.

POSIX.1-2008 uses the word *copy* in several places when discussing buffers. This is not intended to imply implementation.

Historically, *ex* users could not specify numeric buffers because of the ambiguity this would cause; for example, in the command **3 delete 2**, it is unclear whether 2 is a buffer name or a *count*. POSIX.1-2008 requires conformance to historical practice by default, but does not preclude extensions.

Historically, the contents of the unnamed buffer were frequently discarded after commands that did not explicitly affect it; for example, when using the **edit** command to switch files. For consistency and simplicity of specification, POSIX.1-2008 does not permit this behavior.

The *ex* utility did not historically have access to the numeric buffers, and, furthermore, deleting lines in *ex* did not modify their contents. For example, if, after doing a delete in *vi*, the user switched to *ex*, did another delete, and then switched back to *vi*, the contents of the numeric buffers would not have changed. POSIX.1-2008 requires conformance to historical practice. Numeric buffers are described in the *ex* utility in order to confine the description of buffers to a single location in POSIX.1-2008.

The metacharacters that trigger shell expansion in *file* arguments match historical practice, as does the method for doing shell expansion. Implementations wishing to provide users with the flexibility to alter the set of metacharacters are encouraged to provide a **shellmeta** string edit option.

Historically, *ex* commands executed from *vi* refreshed the screen when it did not strictly need to do so; for example, **!date > /dev/null** does not require a screen refresh because the output of the UNIX *date* command requires only a single line of the screen. POSIX.1-2008 requires that the screen be refreshed if it has been overwritten, but makes no requirements as to how an implementation should make that determination. Implementations may prompt and refresh the screen regardless.

Abbreviate

Historical practice was that characters that were entered as part of an abbreviation replacement were subject to **map** expansions, the **showmatch** edit option, further abbreviation expansions, and so on; that is, they were logically pushed onto the terminal input queue, and were not a simple replacement. POSIX.1-2008 requires conformance to historical practice. Historical practice was that whenever a non-word character (that had not been escaped by a <control>-V) was entered after a word character, *vi* would check for abbreviations. The check was based on the type of the character entered before the word character of the word/non-word pair that triggered the check. The word character of the word/non-word pair that triggered the check and all characters entered before the trigger pair that were of that type were included in the check, with the exception of <blank> characters, which always delimited the abbreviation.

This means that, for the abbreviation to work, the *lhs* must end with a word character, there can be no transitions from word to non-word characters (or *vice versa*) other than between the last and next-to-last characters in the *lhs*, and there can be no <blank> characters in the *lhs*. In addition, because of the historical quoting rules, it was impossible to enter a literal <control>-V in the *lhs*. POSIX.1-2008 requires conformance to historical practice. Historical implementations did not inform users when abbreviations that could never be used were entered; implementations are strongly encouraged to do so.

For example, the following abbreviations will work:

```
:ab (p REPLACE
:ab p REPLACE
:ab ( (p REPLACE
```

The following abbreviations will not work:

```
:ab ( REPLACE
:ab (pp REPLACE
```

Historical practice is that words on the *vi* colon command line were subject to abbreviation expansion, including the arguments to the **abbrev** (and more interestingly) the **unabbrev** command. Because there are implementations that do not do abbreviation expansion for the first argument to those commands, this is permitted, but not required, by POSIX.1-2008. However, the following sequence:

```
:ab foo bar
:ab foo baz
```

resulted in the addition of an abbreviation of "baz" for the string "bar" in historical *ex/vi*, and the sequence:

```
:ab fool bar
:ab foo2 bar
:unabbreviate foo2
```

deleted the abbreviation "fool", not "foo2". These behaviors are not permitted by POSIX.1-2008 because they clearly violate the expectations of the user.

It was historical practice that <control>-V, not <backslash>, characters be interpreted as escaping subsequent characters in the **abbreviate** command. POSIX.1-2008 requires conformance to historical practice; however, it should be noted that an abbreviation containing a <blank> will never work.

Append

Historically, any text following a <vertical-line> command separator after an **append**, **change**, or **insert** command became part of the insert text. For example, in the command:

```
:g/pattern/append|stuff1
```

a line containing the text "stuff1" would be appended to each line matching pattern. It was also historically valid to enter:

```
:append|stuff1
stuff2
.
```

and the text on the *ex* command line would be appended along with the text inserted after it. There was an historical bug, however, that the user had to enter two terminating lines (the ' . '

87500 lines) to terminate text input mode in this case. POSIX.1-2008 requires conformance to historical
87501 practice, but disallows the historical need for multiple terminating lines.

87502 **Change**

87503 See the RATIONALE for the **append** command. Historical practice for cursor positioning after
87504 the change command when no text is input, is as described in POSIX.1-2008. However, one
87505 System V implementation is known to have been modified such that the cursor is positioned on
87506 the first address specified, and not on the line before the first address. POSIX.1-2008 disallows
87507 this modification for consistency.

87508 Historically, the **change** command did not support buffer arguments, although some
87509 implementations allow the specification of an optional buffer. This behavior is neither required
87510 nor disallowed by POSIX.1-2008.

87511 **Change Directory**

87512 A common extension in *ex* implementations is to use the elements of a **cdpath** edit option as
87513 prefix directories for *path* arguments to **chdir** that are relative pathnames and that do not have
87514 ' .' or " ." as their first component. Elements in the **cdpath** edit option are <colon>-separated.
87515 The initial value of the **cdpath** edit option is the value of the shell *CDPATH* environment
87516 variable. This feature was not included in POSIX.1-2008 because it does not exist in any of the
87517 implementations considered historical practice.

87518 **Copy**

87519 Historical implementations of *ex* permitted copies to lines inside of the specified range; for
87520 example, **:2,5copy3** was a valid command. POSIX.1-2008 requires conformance to historical
87521 practice.

87522 **Delete**

87523 POSIX.1-2008 requires support for the historical parsing of a **delete** command followed by flags,
87524 without any intervening <blank> characters. For example:

87525 **1dp** Deletes the first line and prints the line that was second.

87526 **1delep** As for **1dp**.

87527 **1d** Deletes the first line, saving it in buffer *p*.

87528 **1d p1l** (Pee-one-ell.) Deletes the first line, saving it in buffer *p*, and listing the line that was
87529 second.

87530 **Edit**

87531 Historically, any *ex* command could be entered as a *+command* argument to the **edit** command,
87532 although some (for example, **insert** and **append**) were known to confuse historical
87533 implementations. For consistency and simplicity of specification, POSIX.1-2008 requires that any
87534 command be supported as an argument to the **edit** command.

87535 Historically, the command argument was executed with the current line set to the last line of the
87536 file, regardless of whether the **edit** command was executed from visual mode or not.
87537 POSIX.1-2008 requires conformance to historical practice.

87538 Historically, the *+command* specified to the **edit** and **next** commands was delimited by the first
87539 <blank>, and there was no way to quote them. For consistency, POSIX.1-2008 requires that the
87540 usual *ex* backslash quoting be provided.

Historically, specifying the *+command* argument to the edit command required a filename to be specified as well; for example, **:edit +100** would always fail. For consistency and simplicity of specification, POSIX.1-2008 does not permit this usage to fail for that reason.

Historically, only the cursor position of the last file edited was remembered by the editor. POSIX.1-2008 requires that this be supported; however, implementations are permitted to remember and restore the cursor position for any file previously edited.

File

Historical versions of the *ex* editor **file** command displayed a current line and number of lines in the edit buffer of 0 when the file was empty, while the *vi* **<control>-G** command displayed a current line and number of lines in the edit buffer of 1 in the same situation. POSIX.1-2008 does not permit this discrepancy, instead requiring that a message be displayed indicating that the file is empty.

Global

The two-pass operation of the **global** and **v** commands is not intended to imply implementation, only the required result of the operation.

The current line and column are set as specified for the individual *ex* commands. This requirement is cumulative; that is, the current line and column must track across all the commands executed by the **global** or **v** commands.

Insert

See the RATIONALE for the **append** command.

Historically, **insert** could not be used with an address of zero; that is, not when the edit buffer was empty. POSIX.1-2008 requires that this command behave consistently with the **append** command.

Join

The action of the **join** command in relation to the special characters is only defined for the POSIX locale because the correct amount of white space after a period varies; in Japanese none is required, in French only a single space, and so on.

List

The historical output of the **list** command was potentially ambiguous. The standard developers believed correcting this to be more important than adhering to historical practice, and POSIX.1-2008 requires unambiguous output.

Map

Historically, command mode maps only applied to command names; for example, if the character 'x' was mapped to 'y', the command **fx** searched for the 'x' character, not the 'y' character. POSIX.1-2008 requires this behavior. Historically, entering **<control>-V** as the first character of a *vi* command was an error. Several implementations have extended the semantics of *vi* such that **<control>-V** means that the subsequent command character is not mapped. This is permitted, but not required, by POSIX.1-2008. Regardless, using **<control>-V** to escape the second or later character in a sequence of characters that might match a **map** command, or any character in text input mode, is historical practice, and stops the entered keys from matching a map. POSIX.1-2008 requires conformance to historical practice.

Historical implementations permitted digits to be used as a **map** command *lhs*, but then ignored the map. POSIX.1-2008 requires that the mapped digits not be ignored.

The historical implementation of the **map** command did not permit **map** commands that were more than a single character in length if the first character was printable. This behavior is permitted, but not required, by POSIX.1-2008.

Historically, mapped characters were remapped unless the **remap** edit option was not set, or the prefix of the mapped characters matched the mapping characters; for example, in the **map**:

```
:map ab abcd
```

the characters "ab" were used as is and were not remapped, but the characters "cd" were mapped if appropriate. This can cause infinite loops in the *vi* mapping mechanisms. POSIX.1-2008 requires conformance to historical practice, and that such loops be interruptible.

Text input maps had the same problems with expanding the *lhs* for the **ex map!** and **unmap!** command as did the **ex abbreviate** and **unabbreviate** commands. See the RATIONALE for the **ex abbreviate** command. POSIX.1-2008 requires similar modification of some historical practice for the **map** and **unmap** commands, as described for the **abbreviate** and **unabbreviate** commands.

Historically, **maps** that were subsets of other **maps** behaved differently depending on the order in which they were defined. For example:

```
:map! ab      short
```

```
:map! abc     long
```

would always translate the characters "ab" to "short", regardless of how fast the characters "abc" were entered. If the entry order was reversed:

```
:map! abc     long
```

```
:map! ab      short
```

the characters "ab" would cause the editor to pause, waiting for the completing 'c' character, and the characters might never be mapped to "short". For consistency and simplicity of specification, POSIX.1-2008 requires that the shortest match be used at all times.

The length of time the editor spends waiting for the characters to complete the *lhs* is unspecified because the timing capabilities of systems are often inexact and variable, and it may depend on other factors such as the speed of the connection. The time should be long enough for the user to be able to complete the sequence, but not long enough for the user to have to wait. Some implementations of *vi* have added a **keytime** option, which permits users to set the number of 0,1 seconds the editor waits for the completing characters. Because mapped terminal function and cursor keys tend to start with an <ESC> character, and <ESC> is the key ending *vi* text input mode, **maps** starting with <ESC> characters are generally exempted from this timeout period, or, at least timed out differently.

Mark

Historically, users were able to set the "previous context" marks explicitly. In addition, the **ex** commands **"** and **'** and the *vi* commands **"**, **'**, **,"**, and **'** all referred to the same mark. In addition, the previous context marks were not set if the command, with which the address setting the mark was associated, failed. POSIX.1-2008 requires conformance to historical practice. Historically, if marked lines were deleted, the mark was also deleted, but would reappear if the change was undone. POSIX.1-2008 requires conformance to historical practice.

The description of the special events that set the **'** and **'** marks matches historical practice. For example, historically the command **/a/** did not set the **'** and **'** marks, but the command

87626 /a/,/b/delete did.

87627 Next

87628 Historically, any *ex* command could be entered as a *+command* argument to the **next** command,
87629 although some (for example, **insert** and **append**) were known to confuse historical
87630 implementations. POSIX.1-2008 requires that any command be permitted and that it behave as
87631 specified. The **next** command can accept more than one file, so usage such as:

87632 `next 'ls [abc] '`

87633 is valid; it need not be valid for the **edit** or **read** commands, for example, because they expect
87634 only one filename.

87635 Historically, the **next** command behaved differently from the **:rewind** command in that it
87636 ignored the force flag if the **autowrite** flag was set. For consistency, POSIX.1-2008 does not
87637 permit this behavior.

87638 Historically, the **next** command positioned the cursor as if the file had never been edited before,
87639 regardless. POSIX.1-2008 does not permit this behavior, for consistency with the **edit** command.

87640 Implementations wanting to provide a counterpart to the **next** command that edited the
87641 previous file have used the command **prev[ious]**, which takes no *file* argument. POSIX.1-2008
87642 does not require this command.

87643 Open

87644 Historically, the **open** command would fail if the **open** edit option was not set. POSIX.1-2008
87645 does not mention the **open** edit option and does not require this behavior. Some historical
87646 implementations do not permit entering open mode from open or visual mode, only from *ex*
87647 mode. For consistency, POSIX.1-2008 does not permit this behavior.

87648 Historically, entering open mode from the command line (that is, *vi +open*) resulted in
87649 anomalous behaviors; for example, the *ex* file and *set* commands, and the *vi* command
87650 <control>-G did not work. For consistency, POSIX.1-2008 does not permit this behavior.

87651 Historically, the **open** command only permitted ' / ' characters to be used as the search pattern
87652 delimiter. For consistency, POSIX.1-2008 requires that the search delimiters used by the **s**, **global**,
87653 and **v** commands be accepted as well.

87654 Preserve

87655 The **preserve** command does not historically cause the file to be considered unmodified for the
87656 purposes of future commands that may exit the editor. POSIX.1-2008 requires conformance to
87657 historical practice.

87658 Historical documentation stated that mail was not sent to the user when preserve was executed;
87659 however, historical implementations did send mail in this case. POSIX.1-2008 requires
87660 conformance to the historical implementations.

Print

The writing of NUL by the **print** command is not specified as a special case because the standard developers did not want to require *ex* to support NUL characters. Historically, characters were displayed using the ARPA standard mappings, which are as follows:

1. Printable characters are left alone.
2. Control characters less than \177 are represented as '^' followed by the character offset from the '@' character in the ASCII map; for example, \007 is represented as '^G'.
3. \177 is represented as '^' followed by '?'.

The display of characters having their eighth bit set was less standard. Existing implementations use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed bytes that had their eighth bit set as the two characters "M-" followed by the seven-bit display as described above.) The latter probably has the best claim to historical practice because it was used for the **-v** option of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

No specific display format is required by POSIX.1-2008.

Explicit dependence on the ASCII character set has been avoided where possible, hence the use of the phrase an "implementation-defined multi-character sequence" for the display of non-printable characters in preference to the historical usage of, for instance, "^I" for the <tab>. Implementations are encouraged to conform to historical practice in the absence of any strong reason to diverge.

Historically, all *ex* commands beginning with the letter 'p' could be entered using capitalized versions of the commands; for example, **P[rint]**, **Pre[serve]**, and **Pu[t]** were all valid command names. POSIX.1-2008 permits, but does not require, this historical practice because capital forms of the commands are used by some implementations for other purposes.

Put

Historically, an *ex* **put** command, executed from open or visual mode, was the same as the open or visual mode **P** command, if the buffer was named and was cut in character mode, and the same as the **p** command if the buffer was named and cut in line mode. If the unnamed buffer was the source of the text, the entire line from which the text was taken was usually **put**, and the buffer was handled as if in line mode, but it was possible to get extremely anomalous behavior. In addition, using the **Q** command to switch into *ex* mode, and then doing a **put** often resulted in errors as well, such as appending text that was unrelated to the (supposed) contents of the buffer. For consistency and simplicity of specification, POSIX.1-2008 does not permit these behaviors. All *ex* **put** commands are required to operate in line mode, and the contents of the buffers are not altered by changing the mode of the editor.

Read

Historically, an *ex* **read** command executed from open or visual mode, executed in an empty file, left an empty line as the first line of the file. For consistency and simplicity of specification, POSIX.1-2008 does not permit this behavior. Historically, a **read** in open or visual mode from a program left the cursor at the last line read in, not the first. For consistency, POSIX.1-2008 does not permit this behavior.

Historical implementations of *ex* were unable to undo **read** commands that read from the output of a program. For consistency, POSIX.1-2008 does not permit this behavior.

Historically, the *ex* and *vi* message after a successful **read** or **write** command specified "characters", not "bytes". POSIX.1-2008 requires that the number of bytes be displayed, not the

number of characters, because it may be difficult in multi-byte implementations to determine the number of characters read. Implementations are encouraged to clarify the message displayed to the user.

Historically, reads were not permitted on files other than type regular, except that FIFO files could be read (probably only because they did not exist when *ex* and *vi* were originally written). Because the historical *ex* evaluated **read!** and **read !** equivalently, there can be no optional way to force the read. POSIX.1-2008 permits, but does not require, this behavior.

Recover

Some historical implementations of the editor permitted users to recover the edit buffer contents from a previous edit session, and then exit without saving those contents (or explicitly discarding them). The intent of POSIX.1-2008 in requiring that the edit buffer be treated as already modified is to prevent this user error.

Rewind

Historical implementations supported the **rewind** command when the user was editing the first file in the list; that is, the file that the **rewind** command would edit. POSIX.1-2008 requires conformance to historical practice.

Substitute

Historically, *ex* accepted an **r** option to the **s** command. The effect of the **r** option was to use the last regular expression used in any command as the pattern, the same as the **~** command. The **r** option is not required by POSIX.1-2008. Historically, the **c** and **g** options were toggled; for example, the command **:s/abc/def/** was the same as **s/abc/def/ccccgggg**. For simplicity of specification, POSIX.1-2008 does not permit this behavior.

The tilde command is often used to replace the last search RE. For example, in the sequence:

```
s/red/blue/
/green
~
```

the **~** command is equivalent to:

```
s/green/blue/
```

Historically, *ex* accepted all of the following forms:

```
s/abc/def/
s/abc/def
s/abc/
s/abc
```

POSIX.1-2008 requires conformance to this historical practice.

The **s** command presumes that the **'^'** character only occupies a single column in the display. Much of the *ex* and *vi* specification presumes that the **<space>** only occupies a single column in the display. There are no known character sets for which this is not true.

Historically, the final column position for the substitute commands was based on previous column movements; a search for a pattern followed by a substitution would leave the column position unchanged, while a **0** command followed by a substitution would change the column position to the first non-**<blank>**. For consistency and simplicity of specification, POSIX.1-2008 requires that the final column position always be set to the first non-**<blank>**.

Set

Historical implementations redisplayed all of the options for each occurrence of the **all** keyword. POSIX.1-2008 permits, but does not require, this behavior.

Tag

No requirement is made as to where *ex* and *vi* shall look for the file referenced by the tag entry. Historical practice has been to look for the path found in the **tags** file, based on the current directory. A useful extension found in some implementations is to look based on the directory containing the tags file that held the entry, as well. No requirement is made as to which reference for the tag in the tags file is used. This is deliberate, in order to permit extensions such as multiple entries in a tags file for a tag.

Because users often specify many different tags files, some of which need not be relevant or exist at any particular time, POSIX.1-2008 requires that error messages about problem tags files be displayed only if the requested tag is not found, and then, only once for each time that the **tag** edit option is changed.

The requirement that the current edit buffer be unmodified is only necessary if the file indicated by the tag entry is not the same as the current file (as defined by the current pathname). Historically, the file would be reloaded if the filename had changed, as well as if the filename was different from the current pathname. For consistency and simplicity of specification, POSIX.1-2008 does not permit this behavior, requiring that the name be the only factor in the decision.

Historically, *vi* only searched for tags in the current file from the current cursor to the end of the file, and therefore, if the **wrapsan** option was not set, tags occurring before the current cursor were not found. POSIX.1-2008 considers this a bug, and implementations are required to search for the first occurrence in the file, regardless.

Undo

The **undo** description deliberately uses the word “modified”. The **undo** command is not intended to undo commands that replace the contents of the edit buffer, such as **edit**, **next**, **tag**, or **recover**.

Cursor positioning after the **undo** command was inconsistent in the historical *vi*, sometimes attempting to restore the original cursor position (**global**, **undo**, and **v** commands), and sometimes, in the presence of maps, placing the cursor on the last line added or changed instead of the first. POSIX.1-2008 requires a simplified behavior for consistency and simplicity of specification.

Version

The **version** command cannot be exactly specified since there is no widely-accepted definition of what the version information should contain. Implementations are encouraged to do something reasonably intelligent.

Write

Historically, the *ex* and *vi* message after a successful **read** or **write** command specified “characters”, not “bytes”. POSIX.1-2008 requires that the number of bytes be displayed, not the number of characters because it may be difficult in multi-byte implementations to determine the number of characters written. Implementations are encouraged to clarify the message displayed to the user.

Implementation-defined tests are permitted so that implementations can make additional checks; for example, for locks or file modification times.

Historically, attempting to append to a nonexistent file caused an error. It has been left unspecified in POSIX.1-2008 to permit implementations to let the **write** succeed, so that the append semantics are similar to those of the historical *csh*.

Historical *vi* permitted empty edit buffers to be written. However, since the way *vi* got around dealing with “empty” files was to always have a line in the edit buffer no matter what, it wrote them as files of a single, empty line. POSIX.1-2008 does not permit this behavior.

Historically, *ex* restored standard output and standard error to their values as of when *ex* was invoked, before writes to programs were performed. This could disturb the terminal configuration as well as be a security issue for some terminals. POSIX.1-2008 does not permit this, requiring that the program output be captured and displayed as if by the *ex* **print** command.

Adjust Window

Historically, the line count was set to the value of the **scroll** option if the type character was end-of-file. This feature was broken on most historical implementations long ago, however, and is not documented anywhere. For this reason, POSIX.1-2008 is resolutely silent.

Historically, the **z** command was <blank>-sensitive and **z +** and **z -** did different things than **z+** and **z-** because the type could not be distinguished from a flag. (The commands **z .** and **z =** were historically invalid.) POSIX.1-2008 requires conformance to this historical practice.

Historically, the **z** command was further <blank>-sensitive in that the *count* could not be <blank>-delimited; for example, the commands **z= 5** and **z- 5** were also invalid. Because the *count* is not ambiguous with respect to either the type character or the flags, this is not permitted by POSIX.1-2008.

Escape

Historically, *ex* filter commands only read the standard output of the commands, letting standard error appear on the terminal as usual. The *vi* utility, however, read both standard output and standard error. POSIX.1-2008 requires the latter behavior for both *ex* and *vi*, for consistency.

Shift Left and Shift Right

Historically, it was possible to add shift characters to increase the effect of the command; for example, <<< outdented (or >>> indented) the lines 3 levels of indentation instead of the default 1. POSIX.1-2008 requires conformance to historical practice.

<control>-D

Historically, the <control>-D command erased the prompt, providing the user with an unbroken presentation of lines from the edit buffer. This is not required by POSIX.1-2008; implementations are encouraged to provide it if possible. Historically, the <control>-D command took, and then ignored, a *count*. POSIX.1-2008 does not permit this behavior.

Write Line Number

Historically, the *ex* = command, when executed in *ex* mode in an empty edit buffer, reported 0, and from open or visual mode, reported 1. For consistency and simplicity of specification, POSIX.1-2008 does not permit this behavior.

Execute

Historically, *ex* did not correctly handle the inclusion of text input commands (that is, **append**, **insert**, and **change**) in executed buffers. POSIX.1-2008 does not permit this exclusion for consistency.

Historically, the logical contents of the buffer being executed did not change if the buffer itself were modified by the commands being executed; that is, buffer execution did not support self-modifying code. POSIX.1-2008 requires conformance to historical practice.

Historically, the @ command took a range of lines, and the @ buffer was executed once per line, with the current line (' . ') set to each specified line. POSIX.1-2008 requires conformance to historical practice.

Some historical implementations did not notice if errors occurred during buffer execution. This, coupled with the ability to specify a range of lines for the *ex* @ command, makes it trivial to cause them to drop **core**. POSIX.1-2008 requires that implementations stop buffer execution if any error occurs, if the specified line doesn't exist, or if the contents of the edit buffer itself are replaced (for example, the buffer executes the *ex* :**edit** command).

Regular Expressions in ex

Historical practice is that the characters in the replacement part of the last **s** command—that is, those matched by entering a ' ~ ' in the regular expression—were not further expanded by the regular expression engine. So, if the characters contained the string "a . , " they would match ' a ' followed by " . , " and not ' a ' followed by any character. POSIX.1-2008 requires conformance to historical practice.

Edit Options in ex

The following paragraphs describe the historical behavior of some edit options that were not, for whatever reason, included in POSIX.1-2008. Implementations are strongly encouraged to only use these names if the functionality described here is fully supported.

extended The **extended** edit option has been used in some implementations of *vi* to provide extended regular expressions instead of basic regular expressions. This option was omitted from POSIX.1-2008 because it is not widespread historical practice.

flash The **flash** edit option historically caused the screen to flash instead of beeping on error. This option was omitted from POSIX.1-2008 because it is not found in some historical implementations.

87863	hardtabs	The hardtabs edit option historically defined the number of columns between hardware tab settings. This option was omitted from POSIX.1-2008 because it was believed to no longer be generally useful.
87864		
87865		
87866	modeline	The modeline (sometimes named modelines) edit option historically caused <i>ex</i> or <i>vi</i> to read the five first and last lines of the file for editor commands. This option is a security problem, and vendors are strongly encouraged to delete it from historical implementations.
87867		
87868		
87869		
87870	open	The open edit option historically disallowed the <i>ex</i> open and visual commands. This edit option was omitted because these commands are required by POSIX.1-2008.
87871		
87872		
87873	optimize	The optimize edit option historically expedited text throughput by setting the terminal to not do automatic <carriage-return> characters when printing more than one logical line of output. This option was omitted from POSIX.1-2008 because it was intended for terminals without addressable cursors, which are rarely, if ever, still used.
87874		
87875		
87876		
87877		
87878	ruler	The ruler edit option has been used in some implementations of <i>vi</i> to present a current row/column ruler for the user. This option was omitted from POSIX.1-2008 because it is not widespread historical practice.
87879		
87880		
87881	sourceany	The sourceany edit option historically caused <i>ex</i> or <i>vi</i> to source start-up files that were owned by users other than the user running the editor. This option is a security problem, and vendors are strongly encouraged to remove it from their implementations.
87882		
87883		
87884		
87885	timeout	The timeout edit option historically enabled the (now standard) feature of only waiting for a short period before returning keys that could be part of a macro. This feature was omitted from POSIX.1-2008 because its behavior is now standard, it is not widely useful, and it was rarely documented.
87886		
87887		
87888		
87889	verbose	The verbose edit option has been used in some implementations of <i>vi</i> to cause <i>vi</i> to output error messages for common errors; for example, attempting to move the cursor past the beginning or end of the line instead of only alerting the screen. (The historical <i>vi</i> only alerted the terminal and presented no message for such errors. The historical editor option terse did not select when to present error messages, it only made existing error messages more or less verbose.) This option was omitted from POSIX.1-2008 because it is not widespread historical practice; however, implementors are encouraged to use it if they wish to provide error messages for naive users.
87890		
87891		
87892		
87893		
87894		
87895		
87896		
87897		
87898	wraplen	The wraplen edit option has been used in some implementations of <i>vi</i> to specify an automatic margin measured from the left margin instead of from the right margin. This is useful when multiple screen sizes are being used to edit a single file. This option was omitted from POSIX.1-2008 because it is not widespread historical practice; however, implementors are encouraged to use it if they add this functionality.
87899		
87900		
87901		
87902		
87903		

autoindent, ai

Historically, the command **0a** did not do any autoindentation, regardless of the current indentation of line 1. POSIX.1-2008 requires that any indentation present in line 1 be used.

autoprint, ap

Historically, the **autoprint** edit option was not completely consistent or based solely on modifications to the edit buffer. Exceptions were the **read** command (when reading from a file, but not from a filter), the **append**, **change**, **insert**, **global**, and **v** commands, all of which were not affected by **autoprint**, and the **tag** command, which was affected by **autoprint**. POSIX.1-2008 requires conformance to historical practice.

Historically, the **autoprint** option only applied to the last of multiple commands entered using <vertical-line> delimiters; for example, **delete** <newline> was affected by **autoprint**, but **delete | version** <newline> was not. POSIX.1-2008 requires conformance to historical practice.

autowrite, aw

Appending the '!' character to the **ex next** command to avoid performing an automatic write was not supported in historical implementations. POSIX.1-2008 requires that the behavior match the other **ex** commands for consistency.

ignorecase, ic

Historical implementations of case-insensitive matching (the **ignorecase** edit option) lead to counter-intuitive situations when uppercase characters were used in range expressions. Historically, the process was as follows:

1. Take a line of text from the edit buffer.
2. Convert uppercase to lowercase in text line.
3. Convert uppercase to lowercase in regular expressions, except in character class specifications.
4. Match regular expressions against text.

This would mean that, with **ignorecase** in effect, the text:

The cat sat on the mat

would be matched by

/^the/

but not by:

/*[A-Z]he/

For consistency with other commands implementing regular expressions, POSIX.1-2008 does not permit this behavior.

paragraphs, para

The ISO POSIX-2:1993 standard made the default **paragraphs** and **sections** edit options implementation-defined, arguing they were historically oriented to the UNIX system *troff* text formatter, and a “portable user” could use the {, }, [[,]], (, and) commands in open or visual mode and have the cursor stop in unexpected places. POSIX.1-2008 specifies their values in the POSIX locale because the unusual grouping (they only work when grouped into two characters at a time) means that they cannot be used for general-purpose movement, regardless.

readonly

Implementations are encouraged to provide the best possible information to the user as to the read-only status of the file, with the exception that they should not consider the current special privileges of the process. This provides users with a safety net because they must force the overwrite of read-only files, even when running with additional privileges.

The **readonly** edit option specification largely conforms to historical practice. The only difference is that historical implementations did not notice that the user had set the **readonly** edit option in cases where the file was already marked read-only for some reason, and would therefore reinitialize the **readonly** edit option the next time the contents of the edit buffer were replaced. This behavior is disallowed by POSIX.1-2008.

report

The requirement that lines copied to a buffer interact differently than deleted lines is historical practice. For example, if the **report** edit option is set to 3, deleting 3 lines will cause a report to be written, but 4 lines must be copied before a report is written.

The requirement that the *ex* **global**, **v**, **open**, **undo**, and **visual** commands present reports based on the total number of lines added or deleted during the command execution, and that commands executed by the **global** and **v** commands not present reports, is historical practice. POSIX.1-2008 extends historical practice by requiring that buffer execution be treated similarly. The reasons for this are two-fold. Historically, only the report by the last command executed from the buffer would be seen by the user, as each new report would overwrite the last. In addition, the standard developers believed that buffer execution had more in common with **global** and **v** commands than it did with other *ex* commands, and should behave similarly, for consistency and simplicity of specification.

showmatch, sm

The length of time the cursor spends on the matching character is unspecified because the timing capabilities of systems are often inexact and variable. The time should be long enough for the user to notice, but not long enough for the user to become annoyed. Some implementations of *vi* have added a **matchtime** option that permits users to set the number of 0,1 second intervals the cursor pauses on the matching character.

showmode

The **showmode** option has been used in some historical implementations of *ex* and *vi* to display the current editing mode when in open or visual mode. The editing modes have generally included “command” and “input”, and sometimes other modes such as “replace” and “change”. The string was usually displayed on the bottom line of the screen at the far right-hand corner. In addition, a preceding ‘*’ character often denoted whether the contents of the edit buffer had been modified. The latter display has sometimes been part of the **showmode** option, and sometimes based on another option. This option was not available in the 4 BSD historical

implementation of *vi*, but was viewed as generally useful, particularly to novice users, and is required by POSIX.1-2008.

The **smd** shorthand for the **showmode** option was not present in all historical implementations of the editor. POSIX.1-2008 requires it, for consistency.

Not all historical implementations of the editor displayed a mode string for command mode, differentiating command mode from text input mode by the absence of a mode string. POSIX.1-2008 permits this behavior for consistency with historical practice, but implementations are encouraged to provide a display string for both modes.

slowopen

Historically, the **slowopen** option was automatically set if the terminal baud rate was less than 1 200 baud, or if the baud rate was 1 200 baud and the **redraw** option was not set. The **slowopen** option had two effects. First, when inserting characters in the middle of a line, characters after the cursor would not be pushed ahead, but would appear to be overwritten. Second, when creating a new line of text, lines after the current line would not be scrolled down, but would appear to be overwritten. In both cases, ending text input mode would cause the screen to be refreshed to match the actual contents of the edit buffer. Finally, terminals that were sufficiently intelligent caused the editor to ignore the **slowopen** option. POSIX.1-2008 permits most historical behavior, extending historical practice to require **slowopen** behaviors if the edit option is set by the user.

tags

The default path for tags files is left unspecified as implementations may have their own **tags** implementations that do not correspond to the historical ones. The default **tags** option value should probably at least include the file **.tags**.

term

Historical implementations of *ex* and *vi* ignored changes to the **term** edit option after the initial terminal information was loaded. This is permitted by POSIX.1-2008; however, implementations are encouraged to permit the user to modify their terminal type at any time.

terse

Historically, the **terse** edit option optionally provided a shorter, less descriptive error message, for some error messages. This is permitted, but not required, by POSIX.1-2008. Historically, most common visual mode errors (for example, trying to move the cursor past the end of a line) did not result in an error message, but simply alerted the terminal. Implementations wishing to provide messages for novice users are urged to do so based on the **edit** option **verbose**, and not **terse**.

window

In historical implementations, the default for the **window** edit option was based on the baud rate as follows:

1. If the baud rate was less than 1 200, the **edit** option **w300** set the window value; for example, the line:

```
set w300=12
```

would set the window option to 12 if the baud rate was less than 1 200.

2. If the baud rate was equal to 1 200, the **edit** option **w1200** set the window value.

3. If the baud rate was greater than 1 200, the **edit** option **w9600** set the window value.

The **w300**, **w1200**, and **w9600** options do not appear in POSIX.1-2008 because of their dependence on specific baud rates.

In historical implementations, the size of the window displayed by various commands was related to, but not necessarily the same as, the **window** edit option. For example, the size of the window was set by the *ex* command **visual 10**, but it did not change the value of the **window** edit option. However, changing the value of the **window** edit option did change the number of lines that were displayed when the screen was repainted. POSIX.1-2008 does not permit this behavior in the interests of consistency and simplicity of specification, and requires that all commands that change the number of lines that are displayed do it by setting the value of the **window** edit option.

wrapmargin, wm

Historically, the **wrapmargin** option did not affect maps inserting characters that also had associated *counts*; for example **:map K 5aABC DEF**. Unfortunately, there are widely used maps that depend on this behavior. For consistency and simplicity of specification, POSIX.1-2008 does not permit this behavior.

Historically, **wrapmargin** was calculated using the column display width of all characters on the screen. For example, an implementation using "**^I**" to represent <tab> characters when the **list** edit option was set, where '**^**' and '**I**' each took up a single column on the screen, would calculate the **wrapmargin** based on a value of 2 for each <tab>. The **number** edit option similarly changed the effective length of the line as well. POSIX.1-2008 requires conformance to historical practice.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this version.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.9.1.1](#) (on page 2317), *ctags*, *ed*, *sed*, *sh*, *stty*, *vi*

[XBD Table 5-1](#) (on page 121), [Chapter 8](#) (on page 173), [Section 9.3](#) (on page 183), [Section 12.2](#) (on page 215)

[XSH *access*](#)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The obsolescent SYNOPSIS is removed, removing the *+command* and *-* options.

- 88061 The following new requirements on POSIX implementations derive from alignment with the
88062 Single UNIX Specification:
- 88063 • In the **map** command description, the sequence *#digit* is added.
 - 88064 • The **directory**, **edcompatible**, **redraw**, and **slowopen** edit options are added.
- 88065 The *ex* utility is extensively changed for alignment with the IEEE P1003.2b draft standard. This
88066 includes changes as a result of the IEEE PASC Interpretations 1003.2 #31, #38, #49, #50, #51, #52,
88067 #55, #56, #57, #61, #62, #63, #64, #65, and #78.
- 88068 The **-l** option is removed.
- 88069 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/23 is applied, correcting a URL.
- 88070 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/8 is applied, making an editorial
88071 correction in the EXTENDED DESCRIPTION.
- 88072 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/9 is applied, removing text describing
88073 behavior on systems with bytes consisting of more than eight bits.
- 88074 **Issue 7**
- 88075 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is
88076 ' _ ' .
- 88077 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.
- 88078 Austin Group Interpretation 1003.1-2001 #121 is applied, clarifying the *ex* **write** command.
- 88079 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 88080 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Utilities

expand**NAME**

expand — convert tabs to spaces

SYNOPSIS

expand *[-t *tablist*] [*file...*]*

DESCRIPTION

The *expand* utility shall write files or the standard input to the standard output with <tab> characters replaced with one or more <space> characters needed to pad to the next tab stop. Any <backspace> characters shall be copied to the output and cause the column position count for tab stop calculations to be decremented; the column position count shall not be decremented below zero.

OPTIONS

The *expand* utility shall conform to XBD [Section 12.2](#) (on page 215).

The following option shall be supported:

-t *tablist* Specify the tab stops. The application shall ensure that the argument *tablist* consists of either a single positive decimal integer or a list of tabstops. If a single number is given, tabs shall be set that number of column positions apart instead of the default 8.

If a list of tabstops is given, the application shall ensure that it consists of a list of two or more positive decimal integers, separated by <blank> or <comma> characters, in ascending order. The <tab> characters shall be set at those specific column positions. Each tab stop *N* shall be an integer value greater than zero, and the list is in strictly ascending order. This is taken to mean that, from the start of a line of output, tabbing to position *N* shall cause the next character output to be in the (*N*+1)th column position on that line.

In the event of *expand* having to process a <tab> at a position beyond the last of those specified in a multiple tab-stop list, the <tab> shall be replaced by a single <space> in the output.

OPERANDS

The following operand shall be supported:

file The pathname of a text file to be used as input.

STDIN

See the INPUT FILES section.

INPUT FILES

Input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *expand*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the processing of <tab> and <space> characters, and for the determination of the width in column positions each character would

expand

Utilities

- 88126 occupy on an output device.
- 88127 *LC_MESSAGES*
- 88128 Determine the locale that should be used to affect the format and contents of
- 88129 diagnostic messages written to standard error.
- 88130 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 88131 **ASYNCHRONOUS EVENTS**
- 88132 Default.
- 88133 **STDOUT**
- 88134 The standard output shall be equivalent to the input files with <tab> characters converted into
- 88135 the appropriate number of <space> characters.
- 88136 **STDERR**
- 88137 The standard error shall be used only for diagnostic messages.
- 88138 **OUTPUT FILES**
- 88139 None.
- 88140 **EXTENDED DESCRIPTION**
- 88141 None.
- 88142 **EXIT STATUS**
- 88143 The following exit values shall be returned:
- 88144 0 Successful completion
- 88145 >0 An error occurred.
- 88146 **CONSEQUENCES OF ERRORS**
- 88147 The *expand* utility shall terminate with an error message and non-zero exit status upon
- 88148 encountering difficulties accessing one of the *file* operands.
- 88149 **APPLICATION USAGE**
- 88150 None.
- 88151 **EXAMPLES**
- 88152 None.
- 88153 **RATIONALE**
- 88154 The *expand* utility is useful for preprocessing text files (before sorting, looking at specific
- 88155 columns, and so on) that contain <tab> characters.
- 88156 See XBD Section 3.103 (on page 50).
- 88157 The *tablist* option-argument consists of integers in ascending order. Utility Syntax Guideline 8
- 88158 mandates that *expand* shall accept the integers (within the single argument) separated using
- 88159 either <comma> or <blank> characters.
- 88160 Earlier versions of this standard allowed the following form in the SYNOPSIS:
- 88161 `expand [-tabstop] [-tab1,tab2,...,tabn] [file ...]`
- 88162 This form is no longer specified by POSIX.1-2008 but may be present in some implementations.
- 88163 **FUTURE DIRECTIONS**
- 88164 None.

88165 **SEE ALSO**88166 *tabs*, *unexpand*88167 XBD [Section 3.103](#) (on page 50), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)88168 **CHANGE HISTORY**

88169 First released in Issue 4.

88170 **Issue 6**

88171 This utility is marked as part of the User Portability Utilities option.

88172 The APPLICATION USAGE section is added.

88173 The obsolescent SYNOPSIS is removed.

88174 The *LC_CTYPE* environment variable description is updated to align with the IEEE P1003.2b
88175 draft standard.

88176 The normative text is reworded to avoid use of the term “must” for application requirements.

88177 **Issue 7**

88178 Austin Group Interpretation 1003.1-2001 #027 is applied.

88179 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

88180 The *expand* utility is moved from the User Portability Utilities option to the Base. User
88181 Portability Utilities is now an option for interactive utilities.

expr

Utilities

88182 **NAME**88183 `expr` — evaluate arguments as an expression88184 **SYNOPSIS**88185 `expr operand...`88186 **DESCRIPTION**88187 The *expr* utility shall evaluate an expression and write the result to standard output.88188 **OPTIONS**

88189 None.

88190 **OPERANDS**

88191 The single expression evaluated by *expr* shall be formed from the *operand* operands, as described
 88192 in the EXTENDED DESCRIPTION section. The application shall ensure that each of the
 88193 expression operator symbols:

88194 `( ) | & = > >= < <= != + - * / % :`88195 and the symbols *integer* and *string* in the table are provided as separate arguments to *expr*.88196 **STDIN**

88197 Not used.

88198 **INPUT FILES**

88199 None.

88200 **ENVIRONMENT VARIABLES**88201 The following environment variables shall affect the execution of *expr*:

88202 **LANG** Provide a default value for the internationalization variables that are unset or null.
 88203 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 88204 variables used to determine the values of locale categories.)

88205 **LC_ALL** If set to a non-empty string value, override the values of all the other
 88206 internationalization variables.

88207 **LC_COLLATE**

88208 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 88209 character collating elements within regular expressions and by the string
 88210 comparison operators.

88211 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 88212 characters (for example, single-byte as opposed to multi-byte characters in
 88213 arguments) and the behavior of character classes within regular expressions.

88214 **LC_MESSAGES**

88215 Determine the locale that should be used to affect the format and contents of
 88216 diagnostic messages written to standard error.

88217 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.88218 **ASYNCHRONOUS EVENTS**

88219 Default.

88220 **STDOUT**

88221 The *expr* utility shall evaluate the expression and write the result, followed by a <newline>, to
 88222 standard output.

Utilities

expr88223 **STDERR**

88224 The standard error shall be used only for diagnostic messages.

88225 **OUTPUT FILES**

88226 None.

88227 **EXTENDED DESCRIPTION**

88228 The formation of the expression to be evaluated is shown in the following table. The symbols
 88229 *expr*, *expr1*, and *expr2* represent expressions formed from *integer* and *string* symbols and the
 88230 expression operator symbols (all separate arguments) by recursive application of the constructs
 88231 described in the table. The expressions are listed in order of increasing precedence, with equal-
 88232 precedence operators grouped between horizontal lines. All of the operators shall be left-
 88233 associative.

Expression	Description
<i>expr1</i> <i>expr2</i>	Returns the evaluation of <i>expr1</i> if it is neither null nor zero; otherwise, returns the evaluation of <i>expr2</i> if it is not null; otherwise, zero.
<i>expr1</i> & <i>expr2</i>	Returns the evaluation of <i>expr1</i> if neither expression evaluates to null or zero; otherwise, returns zero.
<i>expr1</i> = <i>expr2</i> <i>expr1</i> > <i>expr2</i> <i>expr1</i> >= <i>expr2</i> <i>expr1</i> < <i>expr2</i> <i>expr1</i> <= <i>expr2</i> <i>expr1</i> != <i>expr2</i>	Returns the result of a decimal integer comparison if both arguments are integers; otherwise, returns the result of a string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relationship is true, or 0 if the relationship is false. Equal. Greater than. Greater than or equal. Less than. Less than or equal. Not equal.
<i>expr1</i> + <i>expr2</i> <i>expr1</i> - <i>expr2</i>	Addition of decimal integer-valued arguments. Subtraction of decimal integer-valued arguments.
<i>expr1</i> * <i>expr2</i> <i>expr1</i> / <i>expr2</i> <i>expr1</i> % <i>expr2</i>	Multiplication of decimal integer-valued arguments. Integer division of decimal integer-valued arguments, producing an integer result. Remainder of integer division of decimal integer-valued arguments.
<i>expr1</i> : <i>expr2</i>	Matching expression; see below.
(<i>expr</i>)	Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.
<i>integer</i>	An argument consisting only of an (optional) unary minus followed by digits.
<i>string</i>	A string argument; see below.

Matching Expression

The `' : '` matching operator shall compare the string resulting from the evaluation of *expr1* with the regular expression pattern resulting from the evaluation of *expr2*. Regular expression syntax shall be that defined in XBD [Section 9.3](#) (on page 183), except that all patterns are anchored to the beginning of the string (that is, only sequences starting at the first character of a string are matched by the regular expression) and, therefore, it is unspecified whether `' ^ '` is a special character in that context. Usually, the matching operator shall return a string representing the number of characters matched (`' 0 '` on failure). Alternatively, if the pattern contains at least one regular expression subexpression `"[ \ ( . . . \ ) ]"`, the string matched by the back-reference expression `" \1 "` shall be returned. If the back-reference expression `" \1 "` does not match, then the null string shall be returned.

String Operand

A string argument is an argument that cannot be identified as an *integer* argument or as one of the expression operator symbols shown in the OPERANDS section.

The use of string arguments **length**, **substr**, **index**, or **match** produces unspecified results.

EXIT STATUS

The following exit values shall be returned:

- 0 The *expression* evaluates to neither null nor zero.
- 1 The *expression* evaluates to null or zero.
- 2 Invalid *expression*.
- >2 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

After argument processing by the shell, *expr* is not required to be able to tell the difference between an operator and an operand except by the value. If `" $a "` is `' = '`, the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they all may be taken as the `' = '` operator). The following works reliably:

```
expr X$a = X=
```

Also note that this volume of POSIX.1-2008 permits implementations to extend utilities. The *expr* utility permits the integer arguments to be preceded with a unary minus. This means that an integer argument could look like an option. Therefore, the conforming application must employ the `"--"` construct of Guideline 10 of XBD [Section 12.2](#) (on page 215) to protect its operands if there is any chance the first operand might be a negative integer (or any string with a leading minus).

EXAMPLES

The *expr* utility has a rather difficult syntax:

- Many of the operators are also shell control operators or reserved words, so they have to be escaped on the command line.
- Each part of the expression is composed of separate arguments, so liberal usage of <blank> characters is required. For example:

Invalid	Valid
<i>expr</i> 1+2	<i>expr</i> 1 + 2
<i>expr</i> "1 + 2"	<i>expr</i> 1 + 2
<i>expr</i> 1 + (2 * 3)	<i>expr</i> 1 + \(2 * 3 \)

In many cases, the arithmetic and string features provided as part of the shell command language are easier to use than their equivalents in *expr*. Newly written scripts should avoid *expr* in favor of the new features within the shell; see [Section 2.5](#) (on page 2301) and [Section 2.6.4](#) (on page 2310).

The following command:

```
a=$(expr $a + 1)
```

adds 1 to the variable *a*.

The following command, for "*\$a*" equal to either */usr/abc/file* or just *file*:

```
expr $a : '.*\/(.*\)' \| $a
```

returns the last segment of a pathname (that is, *file*). Applications should avoid the character *'/'* used alone as an argument; *expr* may interpret it as the division operator.

The following command:

```
expr "//$a" : '.*\/(.*\)'
```

is a better representation of the previous example. The addition of the *"//"* characters eliminates any ambiguity about the division operator and simplifies the whole expression. Also note that pathnames may contain characters contained in the *IFS* variable and should be quoted to avoid having "*\$a*" expand into multiple arguments.

The following command:

```
expr "$VAR" : '.*'
```

returns the number of characters in *VAR*.

RATIONALE

In an early proposal, EREs were used in the matching expression syntax. This was changed to BREs to avoid breaking historical applications.

The use of a leading <circumflex> in the BRE is unspecified because many historical implementations have treated it as a special character, despite their system documentation. For example:

```
expr foo : ^foo      expr ^foo : ^foo
```

return 3 and 0, respectively, on those systems; their documentation would imply the reverse. Thus, the anchoring condition is left unspecified to avoid breaking historical scripts relying on this undocumented feature.

88343 **FUTURE DIRECTIONS**

88344 None.

88345 **SEE ALSO**88346 [Section 2.5](#) (on page 2301), [Section 2.6.4](#) (on page 2310)88347 XBD [Chapter 8](#) (on page 173), [Section 9.3](#) (on page 183), [Section 12.2](#) (on page 215)88348 **CHANGE HISTORY**

88349 First released in Issue 2.

88350 **Issue 5**

88351 The FUTURE DIRECTIONS section is added.

88352 **Issue 6**88353 The *expr* utility is aligned with the IEEE P1003.2b draft standard, to include resolution of IEEE
88354 PASC Interpretation 1003.2 #104.

88355 The normative text is reworded to avoid use of the term “must” for application requirements.

88356 **Issue 7**

88357 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

88358 The SYNOPSIS and OPERANDS sections are revised to explicitly state that the name of each of
88359 the operands is *operand*.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

*Utilities***false**88360 **NAME**

88361 false — return false value

88362 **SYNOPSIS**

88363 false

88364 **DESCRIPTION**88365 The *false* utility shall return with a non-zero exit code.88366 **OPTIONS**

88367 None.

88368 **OPERANDS**

88369 None.

88370 **STDIN**

88371 Not used.

88372 **INPUT FILES**

88373 None.

88374 **ENVIRONMENT VARIABLES**

88375 None.

88376 **ASYNCHRONOUS EVENTS**

88377 Default.

88378 **STDOUT**

88379 Not used.

88380 **STDERR**

88381 Not used.

88382 **OUTPUT FILES**

88383 None.

88384 **EXTENDED DESCRIPTION**

88385 None.

88386 **EXIT STATUS**88387 The *false* utility shall always exit with a value other than zero.88388 **CONSEQUENCES OF ERRORS**

88389 Default.

88390 **APPLICATION USAGE**

88391 None.

88392 **EXAMPLES**

88393 None.

88394 **RATIONALE**

88395 None.

88396 **FUTURE DIRECTIONS**

88397 None.

88398 **SEE ALSO**88399 *true*

false

Utilities

88400 **CHANGE HISTORY**

88401 First released in Issue 2.

88402 **Issue 6**

88403 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/24 is applied, changing the STDERR
88404 section from “None.” to “Not used.” for alignment with [Section 1.4](#) (on page 2288).

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

fc — process the command history list

SYNOPSIS

```
fc [-r] [-e editor] [first [last]]
fc -l [-nr] [first [last]]
fc -s [old=new] [first]
```

DESCRIPTION

The *fc* utility shall list, or shall edit and re-execute, commands previously entered to an interactive *sh*.

The command history list shall reference commands by number. The first number in the list is selected arbitrarily. The relationship of a number to its command shall not change except when the user logs in and no other process is accessing the list, at which time the system may reset the numbering to start the oldest retained command at another number (usually 1). When the number reaches an implementation-defined upper limit, which shall be no smaller than the value in *HISTSIZE* or 32 767 (whichever is greater), the shell may wrap the numbers, starting the next command with a lower number (usually 1). However, despite this optional wrapping of numbers, *fc* shall maintain the time-ordering sequence of the commands. For example, if four commands in sequence are given the numbers 32 766, 32 767, 1 (wrapped), and 2 as they are executed, command 32 767 is considered the command previous to 1, even though its number is higher.

When commands are edited (when the *-l* option is not specified), the resulting lines shall be entered at the end of the history list and then re-executed by *sh*. The *fc* command that caused the editing shall not be entered into the history list. If the editor returns a non-zero exit status, this shall suppress the entry into the history list and the command re-execution. Any command line variable assignments or redirection operators used with *fc* shall affect both the *fc* command itself as well as the command that results; for example:

```
fc -s -- -l 2>/dev/null
```

reinvokes the previous command, suppressing standard error for both *fc* and the previous command.

OPTIONS

The *fc* utility shall conform to XBD [Section 12.2](#) (on page 215).

The following options shall be supported:

- e editor** Use the editor named by *editor* to edit the commands. The *editor* string is a utility name, subject to search via the *PATH* variable (see XBD [Chapter 8](#), on page 173). The value in the *FCEDIT* variable shall be used as a default when *-e* is not specified. If *FCEDIT* is null or unset, *ed* shall be used as the editor.
- l** (The letter ell.) List the commands rather than invoking an editor on them. The commands shall be written in the sequence indicated by the *first* and *last* operands, as affected by *-r*, with each command preceded by the command number.
- n** Suppress command numbers when listing with *-l*.
- r** Reverse the order of the commands listed (with *-l*) or edited (with neither *-l* nor *-s*).

88447	-s	Re-execute the command without invoking an editor.

88448 OPERANDS

88449 The following operands shall be supported:

88450	<i>first, last</i>	Select the commands to list or edit. The number of previous commands that can be
88451		accessed shall be determined by the value of the <i>HISTSIZE</i> variable. The value of
88452		<i>first</i> or <i>last</i> or both shall be one of the following:

88453	[+]<i>number</i>	A positive number representing a command number; command
88454		numbers can be displayed with the -l option.

88455	<i>-number</i>	A negative decimal number representing the command that was
88456		executed <i>number</i> of commands previously. For example, -1 is the
88457		immediately previous command.

88458 *string* A string indicating the most recently entered command that begins
88459 with that string. If the *old=new* operand is not also specified with *-s*,
88460 the string form of the *first* operand cannot contain an embedded
88461 <equals-sign>.

88462 When the synopsis form with **-s** is used:

- 88463 • If *first* is omitted, the previous command shall be used.

88464 For the synopsis forms without -s:

- If *last* is omitted, *last* shall default to the previous command when *-l* is specified; otherwise, it shall default to *first*.
 - If *first* and *last* are both omitted, the previous 16 commands shall be listed or the previous single command shall be edited (based on the *-l* option).
 - If *first* and *last* are both present, all of the commands from *first* to *last* shall be edited (without *-l*) or listed (with *-l*). Editing multiple commands shall be accomplished by presenting to the editor all of the commands at one time, each command starting on a new line. If *first* represents a newer command than *last*, the commands shall be listed or edited in reverse sequence, equivalent to using *-r*. For example, the following commands on the first line are equivalent to the corresponding commands on the second:

88476 fc -r 10 20 fc 30 40

88477 fc 20 10 fc -r 40 30

- 88478
- 88479
- 88480
- 88481
- When a range of commands is used, it shall not be an error to specify *first* or *last* values that are not in the history list; *fc* shall substitute the value representing the oldest or newest command in the list, as appropriate. For example, if there are only ten commands in the history list, numbered 1 to 10:

88482 fc -1

88483 fc 1 99

88484 shall list and edit, respectively, all ten commands.

88485 *old=new* Replace the first occurrence of string *old* in the commands to be re-executed by the
88486 string *new*.

88487 **STDIN**

88488 Not used.

88489 **INPUT FILES**

88490 None.

88491 **ENVIRONMENT VARIABLES**88492 The following environment variables shall affect the execution of *fc*:

88493 *FCEDIT* This variable, when expanded by the shell, shall determine the default value for
 88494 the *-e editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be
 88495 used as the editor.

88496 *HISTFILE* Determine a pathname naming a command history file. If the *HISTFILE* variable is
 88497 not set, the shell may attempt to access or create a file *.sh_history* in the directory
 88498 referred to by the *HOME* environment variable. If the shell cannot obtain both read
 88499 and write access to, or create, the history file, it shall use an unspecified
 88500 mechanism that allows the history to operate properly. (References to history "file"
 88501 in this section shall be understood to mean this unspecified mechanism in such
 88502 cases.) An implementation may choose to access this variable only when
 88503 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt
 88504 to retrieve entries from, or add entries to, the file, as the result of commands issued
 88505 by the user, the file named by the *ENV* variable, or implementation-defined system
 88506 start-up files. In some historical shells, the history file is initialized just after the
 88507 *ENV* file has been processed. Therefore, it is implementation-defined whether
 88508 changes made to *HISTFILE* after the history file has been initialized are effective.
 88509 Implementations may choose to disable the history list mechanism for users with
 88510 appropriate privileges who do not set *HISTFILE*; the specific circumstances under
 88511 which this occurs are implementation-defined. If more than one instance of the
 88512 shell is using the same history file, it is unspecified how updates to the history file
 88513 from those shells interact. As entries are deleted from the history file, they shall be
 88514 deleted oldest first. It is unspecified when history file entries are physically
 88515 removed from the history file.

88516 *HISTSIZE* Determine a decimal number representing the limit to the number of previous
 88517 commands that are accessible. If this variable is unset, an unspecified default
 88518 greater than or equal to 128 shall be used. The maximum number of commands in
 88519 the history list is unspecified, but shall be at least 128. An implementation may
 88520 choose to access this variable only when initializing the history file, as described
 88521 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*
 88522 after the history file has been initialized are effective.

88523 *LANG* Provide a default value for the internationalization variables that are unset or null.
 88524 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 88525 variables used to determine the values of locale categories.)

88526 *LC_ALL* If set to a non-empty string value, override the values of all the other
 88527 internationalization variables.

88528 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 88529 characters (for example, single-byte as opposed to multi-byte characters in
 88530 arguments and input files).

88531 *LC_MESSAGES*

88532 Determine the locale that should be used to affect the format and contents of
 88533 diagnostic messages written to standard error.

88534 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

88535 ASYNCHRONOUS EVENTS

88536 Default.

88537 STDOUT

88538 When the **-l** option is used to list commands, the format of each command in the list shall be as
88539 follows:

88540 "%d\t%s\n", <line number>, <command>

88541 If both the **-l** and **-n** options are specified, the format of each command shall be:

88542 "\t%s\n", <command>

88543 If the <command> consists of more than one line, the lines after the first shall be displayed as:

88544 "\t%s\n", <continued-command>

88545 STDERR

88546 The standard error shall be used only for diagnostic messages.

88547 OUTPUT FILES

88548 None.

88549 EXTENDED DESCRIPTION

88550 None.

88551 EXIT STATUS

88552 The following exit values shall be returned:

88553 0 Successful completion of the listing.

88554 >0 An error occurred.

88555 Otherwise, the exit status shall be that of the commands executed by *fc*.

88556 CONSEQUENCES OF ERRORS

88557 Default.

88558 APPLICATION USAGE

88559 Since editors sometimes use file descriptors as integral parts of their editing, redirecting their file
88560 descriptors as part of the *fc* command can produce unexpected results. For example, if *vi* is the
88561 *FCEDIT* editor, the command:

88562 *fc -s | more*

88563 does not work correctly on many systems.

88564 Users on windowing systems may want to have separate history files for each window by
88565 setting *HISTFILE* as follows:

88566 *HISTFILE=\$HOME/.sh_hist\$\$*

88567 EXAMPLES

88568 None.

88569 RATIONALE

88570 This utility is based on the *fc* built-in of the KornShell.

88571 An early proposal specified the **-e** option as [**-e** *editor* [*old= new*]], which is not historical
88572 practice. Historical practice in *fc* of either [**-e** *editor*] or [**-e** - [*old= new*]] is acceptable, but not
88573 both together. To clarify this, a new option **-s** was introduced replacing the [**-e** -]. This resolves

the conflict and makes *fc* conform to the Utility Syntax Guidelines.

HISTFILE Some implementations of the KornShell check for the superuser and do not create a history file unless *HISTFILE* is set. This is done primarily to avoid creating unlinked files in the root file system when logging in during single-user mode. *HISTFILE* must be set for the superuser to have history.

HISTSIZE Needed to limit the size of history files. It is the intent of the standard developers that when two shells share the same history file, commands that are entered in one shell shall be accessible by the other shell. Because of the difficulties of synchronization over a network, the exact nature of the interaction is unspecified.

The initialization process for the history file can be dependent on the system start-up files, in that they may contain commands that effectively preempt the settings the user has for *HISTFILE* and *HISTSIZE*. For example, function definition commands are recorded in the history file. If the system administrator includes function definitions in some system start-up file called before the *ENV* file, the history file is initialized before the user can influence its characteristics. In some historical shells, the history file is initialized just after the *ENV* file has been processed. Because of these situations, the text requires the initialization process to be implementation-defined.

Consideration was given to omitting the *fc* utility in favor of the command line editing feature in *sh*. For example, in *vi* editing mode, typing "<ESC> v" is equivalent to:

```
EDITOR=vi fc
```

However, the *fc* utility allows the user the flexibility to edit multiple commands simultaneously (such as *fc 10 20*) and to use editors other than those supported by *sh* for command line editing.

In the KornShell, the alias *r* ("re-do") is preset to *fc -e -* (equivalent to the POSIX *fc -s*). This is probably an easier command name to remember than *fc* ("fix command"), but it does not meet the Utility Syntax Guidelines. Renaming *fc* to *hist* or *redo* was considered, but since this description closely matches historical KornShell practice already, such a renaming was seen as gratuitous. Users are free to create aliases whenever odd historical names such as *fc*, *awk*, *cat*, *grep*, or *yacc* are standardized by POSIX.

Command numbers have no ordering effects; they are like serial numbers. The *-r* option and *-number* operand address the sequence of command execution, regardless of serial numbers. So, for example, if the command number wrapped back to 1 at some arbitrary point, there would be no ambiguity associated with traversing the wrap point. For example, if the command history were:

```
32766: echo 1
32767: echo 2
1: echo 3
```

the number *-2* refers to command 32767 because it is the second previous command, regardless of serial number.

FUTURE DIRECTIONS

None.

SEE ALSO

sh

XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

88616 **CHANGE HISTORY**

88617 First released in Issue 4.

88618 **Issue 5**

88619 The FUTURE DIRECTIONS section is added.

88620 **Issue 6**

88621 This utility is marked as part of the User Portability Utilities option.

88622 In the ENVIRONMENT VARIABLES section, the text “user’s home directory” is updated to
88623 “directory referred to by the *HOME* environment variable”.88624 **Issue 7**

88625 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

fg88626 **NAME**88627 `fg` — run jobs in the foreground88628 **SYNOPSIS**88629 UP `fg [job_id]`88630 **DESCRIPTION**88631 If job control is enabled (see the description of `set -m`), the `fg` utility shall move a background job
88632 from the current environment (see [Section 2.12](#), on page 2331) into the foreground.88633 Using `fg` to place a job into the foreground shall remove its process ID from the list of those
88634 “known in the current shell execution environment”; see [Section 2.9.3.1](#) (on page 2319).88635 **OPTIONS**

88636 None.

88637 **OPERANDS**

88638 The following operand shall be supported:

88639 `job_id` Specify the job to be run as a foreground job. If no `job_id` operand is given, the
88640 `job_id` for the job that was most recently suspended, placed in the background, or
88641 run as a background job shall be used. The format of `job_id` is described in XBD
88642 [Section 3.203](#) (on page 65).88643 **STDIN**

88644 Not used.

88645 **INPUT FILES**

88646 None.

88647 **ENVIRONMENT VARIABLES**88648 The following environment variables shall affect the execution of `fg`:88649 `LANG` Provide a default value for the internationalization variables that are unset or null.
88650 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
88651 variables used to determine the values of locale categories.)88652 `LC_ALL` If set to a non-empty string value, override the values of all the other
88653 internationalization variables.88654 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as
88655 characters (for example, single-byte as opposed to multi-byte characters in
88656 arguments).88657 `LC_MESSAGES`88658 Determine the locale that should be used to affect the format and contents of
88659 diagnostic messages written to standard error.88660 XSI `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.88661 **ASYNCHRONOUS EVENTS**

88662 Default.

88663 **STDOUT**88664 The `fg` utility shall write the command line of the job to standard output in the following format:88665 `"%s\n", <command>`

88666 STDERR

88667 The standard error shall be used only for diagnostic messages.

88668 OUTPUT FILES

88669 None.

88670 EXTENDED DESCRIPTION

88671 None.

88672 EXIT STATUS

88673 The following exit values shall be returned:

88674 0 Successful completion.

88675 >0 An error occurred.

88676 CONSEQUENCES OF ERRORS

88677 If job control is disabled, the *fg* utility shall exit with an error and no job shall be placed in the
88678 foreground.

88679 APPLICATION USAGE

88680 The *fg* utility does not work as expected when it is operating in its own utility execution
88681 environment because that environment has no applicable jobs to manipulate. See the
88682 APPLICATION USAGE section for *bg*. For this reason, *fg* is generally implemented as a shell
88683 regular built-in.

88684 EXAMPLES

88685 None.

88686 RATIONALE

88687 The extensions to the shell specified in this volume of POSIX.1-2008 have mostly been based on
88688 features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also
88689 based on the KornShell. The standard developers examined the characteristics of the C shell
88690 versions of these utilities and found that differences exist. Despite widespread use of the C shell,
88691 the KornShell versions were selected for this volume of POSIX.1-2008 to maintain a degree of
88692 uniformity with the rest of the KornShell features selected (such as the very popular command
88693 line editing features).

88694 FUTURE DIRECTIONS

88695 None.

88696 SEE ALSO

88697 [Section 2.9.3.1](#) (on page 2319), [Section 2.12](#) (on page 2331), *bg*, *kill*, *jobs*, *wait*

88698 XBD [Section 3.203](#) (on page 65), [Chapter 8](#) (on page 173)

88699 CHANGE HISTORY

88700 First released in Issue 4.

88701 Issue 6

88702 This utility is marked as part of the User Portability Utilities option.

88703 The APPLICATION USAGE section is added.

88704 The JC marking is removed from the SYNOPSIS since job control is mandatory in this version.

NAME

file — determine file type

SYNOPSIS

file [-dh] [-M *file*] [-m *file*] *file*...

file -i [-h] *file*...

DESCRIPTION

The *file* utility shall perform a series of tests in sequence on each specified *file* in an attempt to classify it:

1. If *file* does not exist, cannot be read, or its file status could not be determined, the output shall indicate that the file was processed, but that its type could not be determined.
2. If the file is not a regular file, its file type shall be identified. The file types directory, FIFO, socket, block special, and character special shall be identified as such. Other implementation-defined file types may also be identified. If *file* is a symbolic link, by default the link shall be resolved and *file* shall test the type of file referenced by the symbolic link. (See the **-h** and **-i** options below.)
3. If the length of *file* is zero, it shall be identified as an empty file.
4. The *file* utility shall examine an initial segment of *file* and shall make a guess at identifying its contents based on position-sensitive tests. (The answer is not guaranteed to be correct; see the **-d**, **-M**, and **-m** options below.)
5. The *file* utility shall examine *file* and make a guess at identifying its contents based on context-sensitive default system tests. (The answer is not guaranteed to be correct.)
6. The file shall be identified as a data file.

If *file* does not exist, cannot be read, or its file status could not be determined, the output shall indicate that the file was processed, but that its type could not be determined.

If *file* is a symbolic link, by default the link shall be resolved and *file* shall test the type of file referenced by the symbolic link.

OPTIONS

The *file* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the order of the **-m**, **-d**, and **-M** options shall be significant.

The following options shall be supported by the implementation:

- d** Apply any position-sensitive default system tests and context-sensitive default system tests to the file. This is the default if no **-M** or **-m** option is specified.
- h** When a symbolic link is encountered, identify the file as a symbolic link. If **-h** is not specified and *file* is a symbolic link that refers to a nonexistent file, *file* shall identify the file as a symbolic link, as if **-h** had been specified.
- i** If a file is a regular file, do not attempt to classify the type of the file further, but identify the file as specified in the STDOUT section.
- M *file*** Specify the name of a file containing position-sensitive tests that shall be applied to a file in order to classify it (see the EXTENDED DESCRIPTION). No position-sensitive default system tests nor context-sensitive default system tests shall be applied unless the **-d** option is also specified.

file*Utilities*

88746 **-m file** Specify the name of a file containing position-sensitive tests that shall be applied to
88747 a file in order to classify it (see the EXTENDED DESCRIPTION).

88748 If the **-m** option is specified without specifying the **-d** option or the **-M** option, position-
88749 sensitive default system tests shall be applied after the position-sensitive tests specified by the
88750 **-m** option. If the **-M** option is specified with the **-d** option, the **-m** option, or both, or the **-m**
88751 option is specified with the **-d** option, the concatenation of the position-sensitive tests specified
88752 by these options shall be applied in the order specified by the appearance of these options. If a
88753 **-M** or **-m file** option-argument is **-**, the results are unspecified.

OPERANDS

88754 The following operand shall be supported:

88756 *file* A pathname of a file to be tested.

STDIN

88758 The standard input shall be used if a *file* operand is **'-'** and the implementation treats the **'-'**
88759 as meaning standard input. Otherwise, the standard input shall not be used.

INPUT FILES

88761 The *file* can be any file type.

ENVIRONMENT VARIABLES

88763 The following environment variables shall affect the execution of *file*:

88764 **LANG** Provide a default value for the internationalization variables that are unset or null.
88765 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
88766 variables used to determine the values of locale categories.)

88767 **LC_ALL** If set to a non-empty string value, override the values of all the other
88768 internationalization variables.

88769 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
88770 characters (for example, single-byte as opposed to multi-byte characters in
88771 arguments and input files).

88772 **LC_MESSAGES**

88773 Determine the locale that should be used to affect the format and contents of
88774 diagnostic messages written to standard error and informative messages written to
88775 standard output.

88776 **XSI NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

ASYNCHRONOUS EVENTS

88778 Default.

STDOUT

88780 In the POSIX locale, the following format shall be used to identify each operand, *file* specified:

88781 "%s: %s\n", <file>, <type>

88782 The values for <type> are unspecified, except that in the POSIX locale, if *file* is identified as one
88783 of the types listed in the following table, <type> shall contain (but is not limited to) the
88784 corresponding string, unless the file is identified by a position-sensitive test specified by a **-M** or
88785 **-m** option. Each <space> shown in the strings shall be exactly one <space>.

Table 4-9 File Utility Output Strings

If file is:	<type> shall contain the string:	Notes
Nonexistent	cannot open	
Block special	block special	1
Character special	character special	1
Directory	directory	1
FIFO	fifo	1
Socket	socket	1
Symbolic link	symbolic link to	1
Regular file	regular file	1,2
Empty regular file	empty	3
Regular file that cannot be read	cannot open	3
Executable binary	executable	3,4,6
<i>ar</i> archive library (see <i>ar</i>)	archive	3,4,6
Extended <i>cpio</i> format (see <i>pax</i>)	<i>cpio</i> archive	3,4,6
Extended <i>tar</i> format (see ustar in <i>pax</i>)	<i>tar</i> archive	3,4,6
Shell script	commands text	3,5,6
C-language source	c program text	3,5,6
FORTRAN source	fortran program text	3,5,6
Regular file whose type cannot be determined	data	3

Notes:

1. This is a file type test.
2. This test is applied only if the **-i** option is specified.
3. This test is applied only if the **-i** option is not specified.
4. This is a position-sensitive default system test.
5. This is a context-sensitive default system test.
6. Position-sensitive default system tests and context-sensitive default system tests are not applied if the **-M** option is specified unless the **-d** option is also specified.

In the POSIX locale, if *file* is identified as a symbolic link (see the **-h** option), the following alternative output format shall be used:

"%s: %s %s\n", <file>, <type>, <contents of link>"

If the file named by the *file* operand does not exist, cannot be read, or the type of the file named by the *file* operand cannot be determined, this shall not be considered an error that affects the exit status.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

88824 EXTENDED DESCRIPTION

88825 A file specified as an option-argument to the **-m** or **-M** options shall contain one position-
 88826 sensitive test per line, which shall be applied to the file. If the test succeeds, the message field of
 88827 the line shall be printed and no further tests shall be applied, with the exception that tests on
 88828 immediately following lines beginning with a single '**>**' character shall be applied.

88829 Each line shall be composed of the following four <tab>-separated fields. (Implementations may
 88830 allow any combination of one or more white-space characters other than <newline> to act as
 88831 field separators.)

88832 *offset* An unsigned number (optionally preceded by a single '**>**' character) specifying
 88833 the *offset*, in bytes, of the value in the file that is to be compared against the *value*
 88834 field of the line. If the file is shorter than the specified offset, the test shall fail.

88835 If the *offset* begins with the character '**>**', the test contained in the line shall not be
 88836 applied to the file unless the test on the last line for which the *offset* did not begin
 88837 with a '**>**' was successful. By default, the *offset* shall be interpreted as an unsigned
 88838 decimal number. With a leading 0x or 0X, the *offset* shall be interpreted as a
 88839 hexadecimal number; otherwise, with a leading 0, the *offset* shall be interpreted as
 88840 an octal number.

88841 *type* The type of the value in the file to be tested. The type shall consist of the type
 88842 specification characters **d**, **s**, and **u**, specifying signed decimal, string, and
 88843 unsigned decimal, respectively.

88844 The *type* string shall be interpreted as the bytes from the file starting at the
 88845 specified *offset* and including the same number of bytes specified by the *value* field.
 88846 If insufficient bytes remain in the file past the *offset* to match the *value* field, the test
 88847 shall fail.

88848 The type specification characters **d** and **u** can be followed by an optional unsigned
 88849 decimal integer that specifies the number of bytes represented by the type. The
 88850 type specification characters **d** and **u** can be followed by an optional **C**, **S**, **I**, or **L**,
 88851 indicating that the value is of type **char**, **short**, **int**, or **long**, respectively.

88852 The default number of bytes represented by the type specifiers **d**, **f**, and **u** shall
 88853 correspond to their respective C-language types as follows. If the system claims
 88854 conformance to the C-Language Development Utilities option, those specifiers
 88855 shall correspond to the default sizes used in the *c99* utility. Otherwise, the default
 88856 sizes shall be implementation-defined.

88857 For the type specifier characters **d** and **u**, the default number of bytes shall
 88858 correspond to the size of a basic integer type of the implementation. For these
 88859 specifier characters, the implementation shall support values of the optional
 88860 number of bytes to be converted corresponding to the number of bytes in the C-
 88861 language types **char**, **short**, **int**, or **long**. These numbers can also be specified by an
 88862 application as the characters **C**, **S**, **I**, and **L**, respectively. The byte order used when
 88863 interpreting numeric values is implementation-defined, but shall correspond to the
 88864 order in which a constant of the corresponding type is stored in memory on the
 88865 system.

88866 All type specifiers, except for **s**, can be followed by a mask specifier of the form
 88867 **&number**. The mask value shall be AND'ed with the value of the input file before
 88868 the comparison with the *value* field of the line is made. By default, the mask shall
 88869 be interpreted as an unsigned decimal number. With a leading 0x or 0X, the mask
 88870 shall be interpreted as an unsigned hexadecimal number; otherwise, with a leading

Utilities

file

88871		0, the mask shall be interpreted as an unsigned octal number.
88872		The strings byte , short , long , and string shall also be supported as type fields, being interpreted as dC, dS, dL, and s, respectively.
88873		
88874	<i>value</i>	The <i>value</i> to be compared with the value from the file.
88875		If the specifier from the type field is s or string , then interpret the value as a string.
88876		Otherwise, interpret it as a number. If the value is a string, then the test shall succeed only when a string value exactly matches the bytes from the file.
88877		
88878		If the <i>value</i> is a string, it can contain the following sequences:
88879	<i>\character</i>	The <backslash>-escape sequences as specified in XBD Table 5-1 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'). In addition, the escape sequence '\ ' (the <backslash> character followed by a <space> character) shall be recognized to represent a <space> character. The results of using any other character, other than an octal digit, following the <backslash> are unspecified.
88880		
88881		
88882		
88883		
88884		
88885		
88886	<i>\octal</i>	Octal sequences that can be used to represent characters with specific coded values. An octal sequence shall consist of a <backslash> followed by the longest sequence of one, two, or three octal-digit characters (01234567).
88887		
88888		
88889		
88890		By default, any value that is not a string shall be interpreted as a signed decimal number. Any such value, with a leading 0x or 0X, shall be interpreted as an unsigned hexadecimal number; otherwise, with a leading zero, the value shall be interpreted as an unsigned octal number.
88891		
88892		
88893		
88894		If the value is not a string, it can be preceded by a character indicating the comparison to be performed. Permissible characters and the comparisons they specify are as follows:
88895		
88896		
88897	=	The test shall succeed if the value from the file equals the <i>value</i> field.
88898	<	The test shall succeed if the value from the file is less than the <i>value</i> field.
88899	>	The test shall succeed if the value from the file is greater than the <i>value</i> field.
88900	&	The test shall succeed if all of the set bits in the <i>value</i> field are set in the value from the file.
88901		
88902	^	The test shall succeed if at least one of the set bits in the <i>value</i> field is not set in the value from the file.
88903		
88904	x	The test shall succeed if the file is large enough to contain a value of the type specified starting at the offset specified.
88905		
88906	<i>message</i>	The <i>message</i> to be printed if the test succeeds. The <i>message</i> shall be interpreted using the notation for the <i>printf</i> formatting specification; see <i>printf</i> . If the <i>value</i> field was a string, then the value from the file shall be the argument for the <i>printf</i> formatting specification; otherwise, the value from the file shall be the argument.
88907		
88908		
88909		

88910 **EXIT STATUS**

88911 The following exit values shall be returned:

88912 0 Successful completion.

88913 >0 An error occurred.

88914 **CONSEQUENCES OF ERRORS**

88915 Default.

88916 **APPLICATION USAGE**

88917 The *file* utility can only be required to guess at many of the file types because only exhaustive
 88918 testing can determine some types with certainty. For example, binary data on some
 88919 implementations might match the initial segment of an executable or a *tar* archive.

88920 Note that the table indicates that the output contains the stated string. Systems may add text
 88921 before or after the string. For executables, as an example, the machine architecture and various
 88922 facts about how the file was link-edited may be included. Note also that on systems that
 88923 recognize shell script files starting with "#!" as executable files, these may be identified as
 88924 executable binary files rather than as shell scripts.

88925 **EXAMPLES**

88926 Determine whether an argument is a binary executable file:

```
88927 file -- "$1" | grep -q '.*executable' &&  
88928 printf "%s is executable.\n" "$1"
```

88929 **RATIONALE**

88930 The *-f* option was omitted because the same effect can (and should) be obtained using the *xargs*
 88931 utility.

88932 Historical versions of the *file* utility attempt to identify the following types of files: symbolic link,
 88933 directory, character special, block special, socket, *tar* archive, *cpio* archive, SCCS archive, archive
 88934 library, empty, *compress* output, *pack* output, binary data, C source, FORTRAN source, assembler
 88935 source, *nroff*/*troff*/*eqn*/*tbl* source *troff* output, shell script, C shell script, English text, ASCII text,
 88936 various executables, APL workspace, compiled terminfo entries, and CURSES screen images.
 88937 Only those types that are reasonably well specified in POSIX or are directly related to POSIX
 88938 utilities are listed in the table.

88939 Historical systems have used a "magic file" named */etc/magic* to help identify file types. Because
 88940 it is generally useful for users and scripts to be able to identify special file types, the *-m* flag and
 88941 a portable format for user-created magic files has been specified. No requirement is made that an
 88942 implementation of *file* use this method of identifying files, only that users be permitted to add
 88943 their own classifying tests.

88944 In addition, three options have been added to historical practice. The *-d* flag has been added to
 88945 permit users to cause their tests to follow any default system tests. The *-i* flag has been added to
 88946 permit users to test portably for regular files in shell scripts. The *-M* flag has been added to
 88947 permit users to ignore any default system tests.

88948 The POSIX.1-2008 description of default system tests and the interaction between the *-d*, *-M*,
 88949 and *-m* options did not clearly indicate that there were two types of "default system tests". The
 88950 "position-sensitive tests" determine file types by looking for certain string or binary values at
 88951 specific offsets in the file being examined. These position-sensitive tests were implemented in
 88952 historical systems using the magic file described above. Some of these tests are now built into
 88953 the *file* utility itself on some implementations so the output can provide more detail than can be
 88954 provided by magic files. For example, a magic file can easily identify a *core* file on most
 88955 implementations, but cannot name the program file that dropped the core. A magic file could

produce output such as:

```
/home/dwc/core: ELF 32-bit MSB core file SPARC Version 1
```

but by building the test into the *file* utility, you could get output such as:

```
/home/dwc/core: ELF 32-bit MSB core file SPARC Version 1, from 'testprog'
```

These extended built-in tests are still to be treated as position-sensitive default system tests even if they are not listed in */etc/magic* or any other magic file.

The context-sensitive default system tests were always built into the *file* utility. These tests looked for language constructs in text files trying to identify shell scripts, C, FORTRAN, and other computer language source files, and even plain text files. With the addition of the *-m* and *-M* options the distinction between position-sensitive and context-sensitive default system tests became important because the order of testing is important. The context-sensitive system default tests should never be applied before any position-sensitive tests even if the *-d* option is specified before a *-m* option or *-M* option due to the high probability that the context-sensitive system default tests will incorrectly identify arbitrary text files as text files before position-sensitive tests specified by the *-m* or *-M* option would be applied to give a more accurate identification.

Leaving the meaning of *-M* – and *-m* – unspecified allows an existing prototype of these options to continue to work in a backwards-compatible manner. (In that implementation, *-M* – was roughly equivalent to *-d* in POSIX.1-2008.)

The historical *-c* option was omitted as not particularly useful to users or portable shell scripts. In addition, a reasonable implementation of the *file* utility would report any errors found each time the magic file is read.

The historical format of the magic file was the same as that specified by the Rationale in the ISO POSIX-2:1993 standard for the *offset*, *value*, and *message* fields; however, it used less precise type fields than the format specified by the current normative text. The new type field values are a superset of the historical ones.

The following is an example magic file:

```
0 short 070707 cpio archive
0 short 0143561 Byte-swapped cpio archive
0 string 070707 ASCII cpio archive
0 long 0177555 Very old archive
0 short 0177545 Old archive
0 short 017437 Old packed data
0 string \037\036 Packed data
0 string \377\037 Compacted data
0 string \037\235 Compressed data
>2 byte&0x80 >0 Block compressed
>2 byte&0x1f x %d bits
0 string \032\001 Compiled Terminfo Entry
0 short 0433 Curses screen image
0 short 0434 Curses screen image
0 string <ar> System V Release 1 archive
0 string !<arch>\n__.SYMDEF Archive random library
0 string !<arch> Archive
0 string ARF_BEGARF PHIGS clear text archive
0 long 0x137A2950 Scalable OpenFont binary
0 long 0x137A2951 Encrypted scalable OpenFont binary
```

- 89002 The use of a basic integer data type is intended to allow the implementation to choose a word
89003 size commonly used by applications on that architecture.
- 89004 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
89005 but this has been modified in this version.
- 89006 **FUTURE DIRECTIONS**
- 89007 None.
- 89008 **SEE ALSO**
- 89009 *ar, ls, pax, printf*
- 89010 XBD Table 5-1 (on page 121), Chapter 8 (on page 173), Section 12.2 (on page 215)
- 89011 **CHANGE HISTORY**
- 89012 First released in Issue 4.
- 89013 **Issue 6**
- 89014 This utility is marked as part of the User Portability Utilities option.
- 89015 Options and an EXTENDED DESCRIPTION are added as specified in the IEEE P1003.2b draft
89016 standard.
- 89017 IEEE PASC Interpretations 1003.2 #192 and #178 are applied.
- 89018 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/25 is applied, making major changes to
89019 address ambiguities raised in defect reports.
- 89020 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/26 is applied, making it clear in the
89021 OPTIONS section that the **-m**, **-d**, and **-M** options do not comply with Guideline 11 of the
89022 Utility Syntax Guidelines.
- 89023 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/10 is applied, clarifying the specification
89024 characters.
- 89025 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/11 is applied, allowing application
89026 developers to create portable magic files that can match characters in strings, and allowing
89027 common extensions found in existing implementations.
- 89028 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/12 is applied, removing text describing
89029 behavior on systems with bytes consisting of more than eight bits.
- 89030 **Issue 7**
- 89031 Austin Group Interpretation 1003.1-2001 #092 is applied.
- 89032 SD5-XCU-ERN-4 is applied, adding further entries in the Notes column in Table 4-9.
- 89033 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 89034 The *file* utility is moved from the User Portability Utilities option to the Base. User Portability
89035 Utilities is now an option for interactive utilities.
- 89036 The EXAMPLES section is revised to correct an error with the pathname "\$1".

Utilities

find

NAME

find — find files

SYNOPSIS

find [-H|-L] *path*... [*operand_expression*...]

DESCRIPTION

The *find* utility shall recursively descend the directory hierarchy from each file specified by *path*, evaluating a Boolean expression composed of the primaries described in the OPERANDS section for each file encountered. Each *path* operand shall be evaluated unaltered as it was provided, including all trailing <slash> characters; all pathnames for other files encountered in the hierarchy shall consist of the concatenation of the current *path* operand, a <slash> if the current *path* operand did not end in one, and the filename relative to the *path* operand. The relative portion shall contain no dot or dot-dot components, no trailing <slash> characters, and only single <slash> characters between pathname components.

The *find* utility shall be able to descend to arbitrary depths in a file hierarchy and shall not fail due to path length limitations (unless a *path* operand specified by the application exceeds {PATH_MAX} requirements).

The *find* utility shall detect infinite loops; that is, entering a previously visited directory that is an ancestor of the last file encountered. When it detects an infinite loop, *find* shall write a diagnostic message to standard error and shall either recover its position in the hierarchy or terminate.

OPTIONS

The *find* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported by the implementation:

-H Cause the file information and file type evaluated for each symbolic link encountered as a *path* operand on the command line to be those of the file referenced by the link, and not the link itself. If the referenced file does not exist, the file information and type shall be for the link itself. File information and type for symbolic links encountered during the traversal of a file hierarchy shall be that of the link itself.

-L Cause the file information and file type evaluated for each symbolic link encountered as a *path* operand on the command line or encountered during the traversal of a file hierarchy to be those of the file referenced by the link, and not the link itself. If the referenced file does not exist, the file information and type shall be for the link itself.

Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered an error. The last option specified shall determine the behavior of the utility. If neither the **-H** nor the **-L** option is specified, then the file information and type for symbolic links encountered as a *path* operand on the command line or encountered during the traversal of a file hierarchy shall be that of the link itself.

OPERANDS

The following operands shall be supported:

The first operand and subsequent operands up to but not including the first operand that starts with a *' - '*, or is a *' ! '* or a *' ('*, shall be interpreted as *path* operands. If the first operand starts with a *' - '*, or is a *' ! '* or a *' ('*, the behavior is unspecified. Each *path* operand is a pathname of a starting point in the file hierarchy.

The first operand that starts with a *' - '*, or is a *' ! '* or a *' ('*, and all subsequent arguments shall

find*Utilities*

89083 be interpreted as an *expression* made up of the following primaries and operators. In the
 89084 descriptions, wherever *n* is used as a primary argument, it shall be interpreted as a decimal
 89085 integer optionally preceded by a plus ('+') or minus-sign ('-') sign, as follows:

89086 **+*n*** More than *n*.

89087 ***n*** Exactly *n*.

89088 **-*n*** Less than *n*.

89089 The following primaries shall be supported:

89090 **-name *pattern***

89091 The primary shall evaluate as true if the basename of the current pathname
 89092 matches *pattern* using the pattern matching notation described in Section 2.13 (on
 89093 page 2332). The additional rules in Section 2.13.3 (on page 2333) do not apply as
 89094 this is a matching operation, not an expansion.

89095 **-path *pattern***

89096 The primary shall evaluate as true if the current pathname matches *pattern* using
 89097 the pattern matching notation described in Section 2.13 (on page 2332). The
 89098 additional rules in Section 2.13.3 (on page 2333) do not apply as this is a matching
 89099 operation, not an expansion.

89100 **-nouser** The primary shall evaluate as true if the file belongs to a user ID for which the
 89101 *getpwuid()* function defined in the System Interfaces volume of POSIX.1-2008 (or
 89102 equivalent) returns NULL.

89103 **-nogroup** The primary shall evaluate as true if the file belongs to a group ID for which the
 89104 *getgrgid()* function defined in the System Interfaces volume of POSIX.1-2008 (or
 89105 equivalent) returns NULL.

89106 **-xdev** The primary shall always evaluate as true; it shall cause *find* not to continue
 89107 descending past directories that have a different device ID (*st_dev*, see the *stat()*
 89108 function defined in the System Interfaces volume of POSIX.1-2008). If any **-xdev**
 89109 primary is specified, it shall apply to the entire expression even if the **-xdev**
 89110 primary would not normally be evaluated.

89111 **-prune** The primary shall always evaluate as true; it shall cause *find* not to descend the
 89112 current pathname if it is a directory. If the **-depth** primary is specified, the **-prune**
 89113 primary shall have no effect.

89114 **-perm [-]*mode***

89115 The *mode* argument is used to represent file mode bits. It shall be identical in
 89116 format to the *symbolic_mode* operand described in *chmod*, and shall be interpreted
 89117 as follows. To start, a template shall be assumed with all file mode bits cleared. An
 89118 *op* symbol of '+' shall set the appropriate mode bits in the template; '-' shall
 89119 clear the appropriate bits; '=' shall set the appropriate mode bits, without regard
 89120 to the contents of the file mode creation mask of the process. The *op* symbol of '-'
 89121 cannot be the first character of *mode*; this avoids ambiguity with the optional
 89122 leading <hyphen>. Since the initial mode is all bits off, there are not any symbolic
 89123 modes that need to use '-' as the first character.

89124 If the <hyphen> is omitted, the primary shall evaluate as true when the file
 89125 permission bits exactly match the value of the resulting template.

89126 Otherwise, if *mode* is prefixed by a <hyphen>, the primary shall evaluate as true if
 89127 at least all the bits in the resulting template are set in the file permission bits.

Utilities

find

- 89128 **-perm** [-]*onum*
- 89129 If the <hyphen> is omitted, the primary shall evaluate as true when the file mode
- 89130 bits exactly match the value of the octal number *onum* (see the description of the
- 89131 octal *mode* in *chmod*). Otherwise, if *onum* is prefixed by a <hyphen>, the primary
- 89132 shall evaluate as true if at least all of the bits specified in *onum* are set. In both
- 89133 cases, the behavior is unspecified when *onum* exceeds 07777.
- 89134 **-type** *c*
- 89135 The primary shall evaluate as true if the type of the file is *c*, where *c* is 'b', 'c',
- 89136 'd', 'l', 'p', 'f', or 's' for block special file, character special file, directory,
- symbolic link, FIFO, regular file, or socket, respectively.
- 89137 **-links** *n*
- The primary shall evaluate as true if the file has *n* links.
- 89138 **-user** *uname*
- 89139 The primary shall evaluate as true if the file belongs to the user *uname*. If *uname* is
- 89140 a decimal integer and the *getpwnam*() (or equivalent) function does not return a
- valid user name, *uname* shall be interpreted as a user ID.
- 89141 **-group** *gname*
- 89142 The primary shall evaluate as true if the file belongs to the group *gname*. If *gname*
- 89143 is a decimal integer and the *getgrnam*() (or equivalent) function does not return a
- 89144 valid group name, *gname* shall be interpreted as a group ID.
- 89145 **-size** *n*[*c*]
- 89146 The primary shall evaluate as true if the file size in bytes, divided by 512 and
- 89147 rounded up to the next integer, is *n*. If *n* is followed by the character 'c', the size
- shall be in bytes.
- 89148 **-atime** *n*
- 89149 The primary shall evaluate as true if the file access time subtracted from the
- initialization time, divided by 86 400 (with any remainder discarded), is *n*.
- 89150 **-ctime** *n*
- 89151 The primary shall evaluate as true if the time of last change of file status
- 89152 information subtracted from the initialization time, divided by 86 400 (with any
- remainder discarded), is *n*.
- 89153 **-mtime** *n*
- 89154 The primary shall evaluate as true if the file modification time subtracted from the
- initialization time, divided by 86 400 (with any remainder discarded), is *n*.
- 89155 **-exec** *utility_name* [*argument* ...];
- 89156 **-exec** *utility_name* [*argument* ...] {} +
- 89157 The end of the primary expression shall be punctuated by a <semicolon> or by a
- 89158 <plus-sign>. Only a <plus-sign> that immediately follows an argument
- 89159 containing the two characters "{}" shall punctuate the end of the primary
- 89160 expression. Other uses of the <plus-sign> shall not be treated as special.
- 89161 If the primary expression is punctuated by a <semicolon>, the utility *utility_name*
- 89162 shall be invoked once for each pathname and the primary shall evaluate as true if
- 89163 the utility returns a zero value as exit status. A *utility_name* or *argument* containing
- 89164 only the two characters "{}" shall be replaced by the current pathname.
- 89165 If the primary expression is punctuated by a <plus-sign>, the primary shall always
- 89166 evaluate as true, and the pathnames for which the primary is evaluated shall be
- 89167 aggregated into sets. The utility *utility_name* shall be invoked once for each set of
- 89168 aggregated pathnames. Each invocation shall begin after the last pathname in the
- 89169 set is aggregated, and shall be completed before the *find* utility exits and before the
- 89170 first pathname in the next set (if any) is aggregated for this primary, but it is
- 89171 otherwise unspecified whether the invocation occurs before, during, or after the
- 89172 evaluations of other primaries. If any invocation returns a non-zero value as exit
- 89173 status, the *find* utility shall return a non-zero exit status. An argument containing

find*Utilities*

only the two characters "{}" shall be replaced by the set of aggregated pathnames, with each pathname passed as a separate argument to the invoked utility in the same order that it was aggregated. The size of any set of two or more pathnames shall be limited such that execution of the utility does not cause the system's {ARG_MAX} limit to be exceeded. If more than one argument containing only the two characters "{}" is present, the behavior is unspecified.

If a *utility_name* or *argument* string contains the two characters "{}", but not just the two characters "{}", it is implementation-defined whether *find* replaces those two characters or uses the string without change. The current directory for the invocation of *utility_name* shall be the same as the current directory when the *find* utility was started. If the *utility_name* names any of the special built-in utilities (see Section 2.14, on page 2334), the results are undefined.

-ok *utility_name* [*argument* ...];

The **-ok** primary shall be equivalent to **-exec**, except that the use of a <plus-sign> to punctuate the end of the primary expression need not be supported, and *find* shall request affirmation of the invocation of *utility_name* using the current file as an argument by writing to standard error as described in the STDERR section. If the response on standard input is affirmative, the utility shall be invoked. Otherwise, the command shall not be invoked and the value of the **-ok** operand shall be false.

-print The primary shall always evaluate as true; it shall cause the current pathname to be written to standard output.

-newer *file* The primary shall evaluate as true if the modification time of the current file is more recent than the modification time of the file named by the *pathname file*.

-depth The primary shall always evaluate as true; it shall cause descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. If a **-depth** primary is not specified, all entries in a directory shall be acted on after the directory itself. If any **-depth** primary is specified, it shall apply to the entire expression even if the **-depth** primary would not normally be evaluated.

The primaries can be combined using the following operators (in order of decreasing precedence):

(*expression*) True if *expression* is true.

! *expression* Negation of a primary; the unary NOT operator.

expression [-a] *expression*

Conjunction of primaries; the AND operator is implied by the juxtaposition of two primaries or made explicit by the optional **-a** operator. The second expression shall not be evaluated if the first expression is false.

expression -o *expression*

Alternation of primaries; the OR operator. The second expression shall not be evaluated if the first expression is true.

If no *expression* is present, **-print** shall be used as the expression. Otherwise, if the given expression does not contain any of the primaries **-exec**, **-ok**, or **-print**, the given expression shall be effectively replaced by:

(*given_expression*) -print

Utilities

find

- 89219 The **-user**, **-group**, and **-newer** primaries each shall evaluate their respective arguments only
89220 once.
- 89221 When the file type evaluated for the current file is a symbolic link, the results of evaluating the
89222 **-perm** primary are implementation-defined.
- 89223 **STDIN**
- 89224 If the **-ok** primary is used, the response shall be read from the standard input. An entire line
89225 shall be read as the response. Otherwise, the standard input shall not be used.
- 89226 **INPUT FILES**
- 89227 None.
- 89228 **ENVIRONMENT VARIABLES**
- 89229 The following environment variables shall affect the execution of *find*:
- 89230 **LANG** Provide a default value for the internationalization variables that are unset or null.
89231 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
89232 variables used to determine the values of locale categories.)
- 89233 **LC_ALL** If set to a non-empty string value, override the values of all the other
89234 internationalization variables.
- 89235 **LC_COLLATE**
- 89236 Determine the locale for the behavior of ranges, equivalence classes, and multi-
89237 character collating elements used in the pattern matching notation for the **-n**
89238 option and in the extended regular expression defined for the **yesexpr** locale
89239 keyword in the *LC_MESSAGES* category.
- 89240 **LC_CTYPE** This variable determines the locale for the interpretation of sequences of bytes of
89241 text data as characters (for example, single-byte as opposed to multi-byte
89242 characters in arguments), the behavior of character classes within the pattern
89243 matching notation used for the **-n** option, and the behavior of character classes
89244 within regular expressions used in the extended regular expression defined for the
89245 **yesexpr** locale keyword in the *LC_MESSAGES* category.
- 89246 **LC_MESSAGES**
- 89247 Determine the locale used to process affirmative responses, and the locale used to
89248 affect the format and contents of diagnostic messages and prompts written to
89249 standard error.
- 89250 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 89251 **PATH** Determine the location of the *utility_name* for the **-exec** and **-ok** primaries, as
89252 described in XBD [Chapter 8](#) (on page 173).
- 89253 **ASYNCHRONOUS EVENTS**
- 89254 Default.
- 89255 **STDOUT**
- 89256 The **-print** primary shall cause the current pathnames to be written to standard output. The
89257 format shall be:
- 89258 "%s\n", <path>

find*Utilities*89259 **STDERR**

89260 The **-ok** primary shall write a prompt to standard error containing at least the *utility_name* to be
 89261 invoked and the current pathname. In the POSIX locale, the last non-`<blank>` in the prompt shall
 89262 be `' ? '`. The exact format used is unspecified.

89263 Otherwise, the standard error shall be used only for diagnostic messages.

89264 **OUTPUT FILES**

89265 None.

89266 **EXTENDED DESCRIPTION**

89267 None.

89268 **EXIT STATUS**

89269 The following exit values shall be returned:

89270 0 All *path* operands were traversed successfully.

89271 >0 An error occurred.

89272 **CONSEQUENCES OF ERRORS**

89273 Default.

89274 **APPLICATION USAGE**

89275 When used in operands, pattern matching notation, `<semicolon>`, `<left-parenthesis>`, and
 89276 `<right-parenthesis>` characters are special to the shell and must be quoted (see [Section 2.2](#), on
 89277 page 2298).

89278 The bit that is traditionally used for sticky (historically 01000) is specified in the **-perm** primary
 89279 using the octal number argument form. Since this bit is not defined by this volume of
 89280 POSIX.1-2008, applications must not assume that it actually refers to the traditional sticky bit.

89281 **EXAMPLES**

89282 1. The following commands are equivalent:

89283 `find .`

89284 `find . -print`

89285 They both write out the entire directory hierarchy from the current directory.

89286 2. The following command:

89287 `find / \( -name tmp -o -name '*.xx' \) -atime +7 -exec rm {} \;`

89288 removes all files named **tmp** or ending in **.xx** that have not been accessed for seven or
 89289 more 24-hour periods.

89290 3. The following command:

89291 `find . -perm -o+w,+s`

89292 prints (**-print** is assumed) the names of all files in or below the current directory, with all
 89293 of the file permission bits **S_ISUID**, **S_ISGID**, and **S_IWOTH** set.

89294 4. The following command:

89295 `find . -name SCCS -prune -o -print`

89296 recursively prints pathnames of all files in the current directory and below, but skips
 89297 directories named **SCCS** and files in them.

5. The following command:

```
find . -print -name SCCS -prune
```

behaves as in the previous example, but prints the names of the SCCS directories.

6. The following command is roughly equivalent to the **-nt** extension to *test*:

```
if [ -n "$(find file1 -prune -newer file2)" ]; then
    printf %s\\n "file1 is newer than file2"
fi
```

7. The descriptions of **-atime**, **-ctime**, and **-mtime** use the terminology *n* “86 400 second periods (days)”. For example, a file accessed at 23:59 is selected by:

```
find . -atime -1 -print
```

at 00:01 the next day (less than 24 hours later, not more than one day ago); the midnight boundary between days has no effect on the 24-hour calculation.

8. The following command:

```
find . ! -name . -prune -name '*.old' -exec \
    sh -c 'mv "$@" ../old/' sh {} +
```

performs the same task as:

```
mv ./*.old ../*.old ../old/
```

while avoiding an “Argument list too long” error if there are a large number of files ending with **.old** (and avoiding “No such file or directory” errors if no files match **./*.old** or **./*.old**).

The alternative:

```
find . ! -name . -prune -name '*.old' -exec mv {} ../old/ \;
```

is less efficient if there are many files to move because it executes one *mv* command per file.

9. On systems configured to mount removable media on directories under **/media**, the following command searches the file hierarchy for files larger than 100 000 KB without searching any mounted removable media:

```
find / -path /media -prune -o -size +200000 -print
```

10. Except for the root directory, and **"/"** on implementations where **"/"** does not refer to the root directory, no pattern given to **-name** will match a **<slash>**, because trailing **<slash>** characters are ignored when computing the basename of the file under evaluation. Given two empty directories named **foo** and **bar**, the following command:

```
find foo/// bar/// -name foo -o -name 'bar?*'
```

prints only the line **foo///**.

RATIONALE

The **-a** operator was retained as an optional operator for compatibility with historical shell scripts, even though it is redundant with expression concatenation.

The descriptions of the **'-'** modifier on the *mode* and *onum* arguments to the **-perm** primary agree with historical practice on BSD and System V implementations. System V and BSD documentation both describe it in terms of checking additional bits; in fact, it uses the same bits, but checks for having at least all of the matching bits set instead of having exactly the matching

find*Utilities*

bits set.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because:

- Implementations may desire more descriptive prompts than those used on historical implementations.
- Since the historical prompt strings do not terminate with <newline> characters, there is no portable way for another program to interact with the prompts of this utility via pipes.

Therefore, an application using this prompting option relies on the system to provide the most suitable dialog directly with the user, based on the general guidelines specified.

The **-name** *file* operand was changed to use the shell pattern matching notation so that *find* is consistent with other utilities using pattern matching.

The **-size** operand refers to the size of a file, rather than the number of blocks it may occupy in the file system. The intent is that the *st_size* field defined in the System Interfaces volume of POSIX.1-2008 should be used, not the *st_blocks* found in historical implementations. There are at least two reasons for this:

1. In both System V and BSD, *find* only uses *st_size* in size calculations for the operands specified by this volume of POSIX.1-2008. (BSD uses *st_blocks* only when processing the **-ls** primary.)
2. Users usually think of file size in terms of bytes, which is also the unit used by the *ls* utility for the output from the **-l** option. (In both System V and BSD, *ls* uses *st_size* for the **-l** option size field and uses *st_blocks* for the *ls -s* calculations. This volume of POSIX.1-2008 does not specify *ls -s*.)

The descriptions of **-atime**, **-ctime**, and **-mtime** were changed from the SVID description of *n* "days" to *n* being the result of the integer division of the time difference in seconds by 86 400. The description is also different in terms of the exact timeframe for the *n* case (*versus* the *+n* or *-n*), but it matches all known historical implementations. It refers to one 86 400 second period in the past, not any time from the beginning of that period to the current time. For example, **-atime 2** is true if the file was accessed any time in the period from 72 hours to 48 hours ago.

Historical implementations do not modify "**{ }**" when it appears as a substring of an **-exec** or **-ok** *utility_name* or argument string. There have been numerous user requests for this extension, so this volume of POSIX.1-2008 allows the desired behavior. At least one recent implementation does support this feature, but encountered several problems in managing memory allocation and dealing with multiple occurrences of "**{ }**" in a string while it was being developed, so it is not yet required behavior.

Assuming the presence of **-print** was added to correct a historical pitfall that plagues novice users, it is entirely upwards-compatible from the historical System V *find* utility. In its simplest form (*find directory*), it could be confused with the historical BSD fast *find*. The BSD developers agreed that adding **-print** as a default expression was the correct decision and have added the fast *find* functionality within a new utility called *locate*.

Historically, the **-L** option was implemented using the primary **-follow**. The **-H** and **-L** options were added for two reasons. First, they offer a finer granularity of control and consistency with other programs that walk file hierarchies. Second, the **-follow** primary always evaluated to true. As they were historically really global variables that took effect before the traversal began, some valid expressions had unexpected results. An example is the expression **-print -o -follow**. Because **-print** always evaluates to true, the standard order of evaluation implies that **-follow** would never be evaluated. This was never the case. Historical practice for the **-follow** primary,

however, is not consistent. Some implementations always follow symbolic links on the command line whether **-follow** is specified or not. Others follow symbolic links on the command line only if **-follow** is specified. Both behaviors are provided by the **-H** and **-L** options, but scripts using the current **-follow** primary would be broken if the **-follow** option is specified to work either way.

Since the **-L** option resolves all symbolic links and the **-type l** primary is true for symbolic links that still exist after symbolic links have been resolved, the command:

```
find -L . -type l
```

prints a list of symbolic links reachable from the current directory that do not resolve to accessible files.

A feature of SVR4's *find* utility was the **-exec** primary's **+** terminator. This allowed filenames containing special characters (especially <newline> characters) to be grouped together without the problems that occur if such filenames are piped to *xargs*. Other implementations have added other ways to get around this problem, notably a **-print0** primary that wrote filenames with a null byte terminator. This was considered here, but not adopted. Using a null terminator meant that any utility that was going to process *find*'s **-print0** output had to add a new option to parse the null terminators it would now be reading.

The "**-exec** ... { } +" syntax adopted was a result of IEEE PASC Interpretation 1003.2 #210. It should be noted that this is an incompatible change to the ISO/IEC 9899:1999 standard. For example, the following command prints all files with a '-' after their name if they are regular files, and a '+' otherwise:

```
find / -type f -exec echo { } - ';' \; -o -exec echo { } + ';' \;
```

The change invalidates usage like this. Even though the previous standard stated that this usage would work, in practice many did not support it and the standard developers felt it better to now state that this was not allowable.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.2 (on page 2298), Section 2.13 (on page 2332), Section 2.14 (on page 2334), *chmod*, *pax*, *sh*, *test*

XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

XSH *fstatat()*, *getrgid()*, *getpwuid()*

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **-perm [-]onum** primary is supported.

The *find* utility is aligned with the IEEE P1003.2b draft standard, to include processing of symbolic links and changes to the description of the **atime**, **ctime**, and **mtime** operands.

IEEE PASC Interpretation 1003.2 #210 is applied, extending the **-exec** operand.

- 89428 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/13 is applied, updating the RATIONALE
89429 section to be consistent with the normative text.
- 89430 **Issue 7**
- 89431 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the
89432 *LC_MESSAGES* environment variable.
- 89433 Austin Group Interpretation 1003.1-2001 #127 is applied, rephrasing the description of the **-exec**
89434 primary to be “immediately follows”.
- 89435 Austin Group Interpretation 1003.1-2001 #185 is applied, clarifying the requirements for the **-H**
89436 and **-L** options.
- 89437 Austin Group Interpretation 1003.1-2001 #186 is applied, clarifying the requirements for the
89438 evaluation of *path* operands.
- 89439 Austin Group Interpretation 1003.1-2001 #195 is applied, clarifying the interpretation of the first
89440 operand.
- 89441 SD5-XCU-ERN-48 is applied, clarifying the **-L** option in the case that the referenced file does not
89442 exist.
- 89443 SD5-XCU-ERN-89 is applied, updating the OPERANDS section.
- 89444 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 89445 SD5-XCU-ERN-117 is applied, clarifying the **-perm** operand.
- 89446 SD5-XCU-ERN-122 is applied, adding a new EXAMPLE.
- 89447 The description of the **-name** primary is revised and the **-path** primary is added (with a new
89448 example).

Utilities

fold

NAME

fold — filter for folding lines

SYNOPSIS

fold [-bs] [-w *width*] [*file...*]

DESCRIPTION

The *fold* utility is a filter that shall fold lines from its input files, breaking the lines to have a maximum of *width* column positions (or bytes, if the **-b** option is specified). Lines shall be broken by the insertion of a <newline> such that each output line (referred to later in this section as a *segment*) is the maximum width possible that does not exceed the specified number of column positions (or bytes). A line shall not be broken in the middle of a character. The behavior is undefined if *width* is less than the number of columns any single character in the input would occupy.

If the <carriage-return>, <backspace>, or <tab> characters are encountered in the input, and the **-b** option is not specified, they shall be treated specially:

<backspace> The current count of line width shall be decremented by one, although the count never shall become negative. The *fold* utility shall not insert a <newline> immediately before or after any <backspace>, unless the following character has a width greater than 1 and would cause the line width to exceed *width*.

<carriage-return>

The current count of line width shall be set to zero. The *fold* utility shall not insert a <newline> immediately before or after any <carriage-return>.

<tab>

Each <tab> encountered shall advance the column position pointer to the next tab stop. Tab stops shall be at each column position *n* such that *n* modulo 8 equals 1.

OPTIONS

The *fold* utility shall conform to XBD [Section 12.2](#) (on page 215).

The following options shall be supported:

-b Count *width* in bytes rather than column positions.

-s If a segment of a line contains a <blank> within the first *width* column positions (or bytes), break the line after the last such <blank> meeting the width constraints. If there is no <blank> meeting the requirements, the **-s** option shall have no effect for that output segment of the input line.

-w *width* Specify the maximum line length, in column positions (or bytes if **-b** is specified). The results are unspecified if *width* is not a positive decimal number. The default value shall be 80.

OPERANDS

The following operand shall be supported:

file A pathname of a text file to be folded. If no *file* operands are specified, the standard input shall be used.

STDIN

The standard input shall be used if no *file* operands are specified, and shall be used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

fold*Utilities*89491 **INPUT FILES**

89492 If the **-b** option is specified, the input files shall be text files except that the lines are not limited
 89493 to {LINE_MAX} bytes in length. If the **-b** option is not specified, the input files shall be text files.

89494 **ENVIRONMENT VARIABLES**

89495 The following environment variables shall affect the execution of *fold*:

89496 **LANG** Provide a default value for the internationalization variables that are unset or null.
 89497 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 89498 variables used to determine the values of locale categories.)

89499 **LC_ALL** If set to a non-empty string value, override the values of all the other
 89500 internationalization variables.

89501 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 89502 characters (for example, single-byte as opposed to multi-byte characters in
 89503 arguments and input files), and for the determination of the width in column
 89504 positions each character would occupy on a constant-width font output device.

89505 **LC_MESSAGES**

89506 Determine the locale that should be used to affect the format and contents of
 89507 diagnostic messages written to standard error.

89508 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

89509 **ASYNCHRONOUS EVENTS**

89510 Default.

89511 **STDOUT**

89512 The standard output shall be a file containing a sequence of characters whose order shall be
 89513 preserved from the input files, possibly with inserted <newline> characters.

89514 **STDERR**

89515 The standard error shall be used only for diagnostic messages.

89516 **OUTPUT FILES**

89517 None.

89518 **EXTENDED DESCRIPTION**

89519 None.

89520 **EXIT STATUS**

89521 The following exit values shall be returned:

89522 0 All input files were processed successfully.

89523 >0 An error occurred.

89524 **CONSEQUENCES OF ERRORS**

89525 Default.

APPLICATION USAGE

The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths. The *cut* utility should be used when the number of lines (or records) needs to remain constant. The *fold* utility should be used when the contents of long lines need to be kept contiguous.

The *fold* utility is frequently used to send text files to printers that truncate, rather than fold, lines wider than the printer is able to print (usually 80 or 132 column positions).

EXAMPLES

An example invocation that submits a file of possibly long lines to the printer (under the assumption that the user knows the line width of the printer to be assigned by *lp*):

```
fold -w 132 bigfile | lp
```

RATIONALE

Although terminal input in canonical processing mode requires the erase character (frequently set to <backspace>) to erase the previous character (not byte or column position), terminal output is not buffered and is extremely difficult, if not impossible, to parse correctly; the interpretation depends entirely on the physical device that actually displays/prints/stores the output. In all known internationalized implementations, the utilities producing output for mixed column-width output assume that a <backspace> character backs up one column position and outputs enough <backspace> characters to return to the start of the character when <backspace> is used to provide local line motions to support underlining and boldening operations. Since *fold* without the *-b* option is dealing with these same constraints, <backspace> is always treated as backing up one column position rather than backing up one character.

Historical versions of the *fold* utility assumed 1 byte was one character and occupied one column position when written out. This is no longer always true. Since the most common usage of *fold* is believed to be folding long lines for output to limited-length output devices, this capability was preserved as the default case. The *-b* option was added so that applications could *fold* files with arbitrary length lines into text files that could then be processed by the standard utilities. Note that although the width for the *-b* option is in bytes, a line is never split in the middle of a character. (It is unspecified what happens if a width is specified that is too small to hold a single character found in the input followed by a <newline>.)

The tab stops are hardcoded to be every eighth column to meet historical practice. No new method of specifying other tab stops was invented.

FUTURE DIRECTIONS

None.

SEE ALSO

cut

XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 4.

Issue 6

The normative text is reworded to avoid use of the term “must” for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #092 is applied.

fold*Utilities*

89568	Austin Group Interpretation 1003.1-2001 #204 is applied, updating the DESCRIPTION to clarify
89569	when a <newline> can be inserted before or after a <backspace>.
89570	SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

fort77 — FORTRAN compiler (**F**ORTRAN)

SYNOPSIS

```
FD fort77 [-c] [-g] [-L directory]... [-O optlevel] [-o outfile] [-s]
      [-w] operand...
```

DESCRIPTION

The *fort77* utility is the interface to the FORTRAN compilation system; it shall accept the full FORTRAN-77 language defined by the ANSI X3.9-1978 standard. The system conceptually consists of a compiler and link editor. The files referenced by *operands* are compiled and linked to produce an executable file. It is unspecified whether the linking occurs entirely within the operation of *fort77*; some implementations may produce objects that are not fully resolved until the file is executed.

If the *-c* option is present, for all pathname operands of the form *file.f*, the files:

```
$(basename pathname.f).o
```

shall be created or overwritten as the result of successful compilation. If the *-c* option is not specified, it is unspecified whether such *.o* files are created or deleted for the *file.f* operands.

If there are no options that prevent link editing (such as *-c*) and all operands compile and link without error, the resulting executable file shall be written into the file named by the *-o* option (if present) or to the file **a.out**. The executable file shall be created as specified in the System Interfaces volume of POSIX.1-2008, except that the file permissions shall be set to:

```
S_IRWXO | S_IRWXG | S_IRWXU
```

and that the bits specified by the *umask* of the process shall be cleared.

OPTIONS

The *fort77* utility shall conform to XBD [Section 12.2](#) (on page 215), except that:

- The *-l library* operands have the format of options, but their position within a list of operands affects the order in which libraries are searched.
- The order of specifying the multiple *-L* options is significant.
- Conforming applications shall specify each option separately; that is, grouping option letters (for example, *-cg*) need not be recognized by all implementations.

The following options shall be supported:

<i>-c</i> <i>-g</i> <i>-s</i> <i>-o outfile</i>	Suppress the link-edit phase of the compilation, and do not remove any object files that are produced. Produce symbolic information in the object or executable files; the nature of this information is unspecified, and may be modified by implementation-defined interactions with other options. Produce object or executable files, or both, from which symbolic and other information not required for proper execution using the <i>exec</i> family of functions defined in the System Interfaces volume of POSIX.1-2008 has been removed (stripped). If both <i>-g</i> and <i>-s</i> options are present, the action taken is unspecified. Use the pathname <i>outfile</i>, instead of the default a.out, for the executable file produced. If the <i>-o</i> option is present with <i>-c</i>, the result is unspecified.

fort77*Utilities*

- 89612 **-L *directory*** Change the algorithm of searching for the libraries named in **-I** operands to look in
 89613 the directory named by the *directory* pathname before looking in the usual places.
 89614 Directories named in **-L** options shall be searched in the specified order. At least
 89615 ten instances of this option shall be supported in a single *fort77* command
 89616 invocation. If a directory specified by a **-L** option contains a file named **libf.a**, the
 89617 results are unspecified.
- 89618 **-O *optlevel*** Specify the level of code optimization. If the *optlevel* option-argument is the digit
 89619 '0', all special code optimizations shall be disabled. If it is the digit '1', the
 89620 nature of the optimization is unspecified. If the **-O** option is omitted, the nature of
 89621 the system's default optimization is unspecified. It is unspecified whether code
 89622 generated in the presence of the **-O 0** option is the same as that generated when
 89623 **-O** is omitted. Other *optlevel* values may be supported.
- 89624 **-w** Suppress warnings.
- 89625 Multiple instances of **-L** options can be specified.

OPERANDS

- 89626 An *operand* is either in the form of a pathname or the form **-I *library***. At least one operand of the
 89627 pathname form shall be specified. The following operands shall be supported:
- 89628 ***file.f*** The pathname of a FORTRAN source file to be compiled and optionally passed to
 89629 the link editor. The filename operand shall be of this form if the **-c** option is used.
- 89630 ***file.a*** A library of object files typically produced by *ar*, and passed directly to the link
 89631 editor. Implementations may recognize implementation-defined suffixes other
 89632 than **.a** as denoting object file libraries.
- 89633 ***file.o*** An object file produced by *fort77 -c* and passed directly to the link editor.
 89634 Implementations may recognize implementation-defined suffixes other than **.o** as
 89635 denoting object files.
- 89636 The processing of other files is implementation-defined.
- 89637 **-I *library*** (The letter ell.) Search the library named:
- 89638 *liblibrary.a*
- 89639 A library is searched when its name is encountered, so the placement of a **-I**
 89640 operand is significant. Several standard libraries can be specified in this manner, as
 89641 described in the EXTENDED DESCRIPTION section. Implementations may
 89642 recognize implementation-defined suffixes other than **.a** as denoting libraries.

STDIN

- 89644 Not used.

INPUT FILES

- 89646 The input file shall be one of the following: a text file containing FORTRAN source code; an
 89647 object file in the format produced by *fort77 -c*; or a library of object files, in the format produced
 89648 by archiving zero or more object files, using *ar*. Implementations may supply additional utilities
 89649 that produce files in these formats. Additional input files are implementation-defined.
- 89650 A <tab> encountered within the first six characters on a line of source code shall cause the
 89651 compiler to interpret the following character as if it were the seventh character on the line (that
 89652 is, in column 7).

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *fort77*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NSI NLSPATH Determine the location of message catalogs for the processing of **LC_MESSAGES**.

TMPDIR Determine the pathname that should override the default directory for temporary files, if any.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages. If more than one *file* operand ending in *.f* (or possibly other unspecified suffixes) is given, for each such file:

"%s:\n", <*file*>

may be written to allow identification of the diagnostic message with the appropriate input file.

This utility may produce warning messages about certain conditions that do not warrant returning an error (non-zero) exit value.

OUTPUT FILES

Object files, listing files, and executable files shall be produced in unspecified formats.

EXTENDED DESCRIPTION**Standard Libraries**

The *fort77* utility shall recognize the following **-l** operand for the standard library:

-lf This library contains all functions referenced in the ANSI X3.9-1978 standard. This operand shall not be required to be present to cause a search of this library.

In the absence of options that inhibit invocation of the link editor, such as **-c**, the *fort77* utility shall cause the equivalent of a **-lf** operand to be passed to the link editor as the last **-l** operand, causing it to be searched after all other object files and libraries are loaded.

It is unspecified whether the library **libf.a** exists as a regular file. The implementation may accept as **-l** operands names of objects that do not exist as regular files.

89693 **External Symbols**

89694 The FORTRAN compiler and link editor shall support the significance of external symbols up to
 89695 a length of at least 31 bytes; case folding is permitted. The action taken upon encountering
 89696 symbols exceeding the implementation-defined maximum symbol length is unspecified.

89697 The compiler and link editor shall support a minimum of 511 external symbols per source or
 89698 object file, and a minimum of 4 095 external symbols total. A diagnostic message is written to
 89699 standard output if the implementation-defined limit is exceeded; other actions are unspecified.

89700 **EXIT STATUS**

89701 The following exit values shall be returned:

89702 0 Successful compilation or link edit.

89703 >0 An error occurred.

89704 **CONSEQUENCES OF ERRORS**

89705 When *fort77* encounters a compilation error, it shall write a diagnostic to standard error and
 89706 continue to compile other source code operands. It shall return a non-zero exit status, but it is
 89707 implementation-defined whether an object module is created. If the link edit is unsuccessful, a
 89708 diagnostic message shall be written to standard error, and *fort77* shall exit with a non-zero status.

89709 **APPLICATION USAGE**

89710 None.

89711 **EXAMPLES**

89712 The following usage example compiles **xyz.f** and creates the executable file **foo**:

89713 `fort77 -o foo xyz.f`

89714 The following example compiles **xyz.f** and creates the object file **xyz.o**:

89715 `fort77 -c xyz.f`

89716 The following example compiles **xyz.f** and creates the executable file **a.out**:

89717 `fort77 xyz.f`

89718 The following example compiles **xyz.f**, links it with **b.o**, and creates the executable **a.out**:

89719 `fort77 xyz.f b.o`

89720 **RATIONALE**

89721 The name of this utility was chosen as *fort77* to parallel the renaming of the C compiler. The
 89722 name *f77* was not chosen to avoid problems with historical implementations. The
 89723 ANSI X3.9-1978 standard was selected as a normative reference because the ISO/IEC version of
 89724 FORTRAN-77 has been superseded by the ISO/IEC 1539: 1991 standard.

89725 The file inclusion and symbol definition **#define** mechanisms used by the *c99* utility were not
 89726 included in this volume of POSIX.1-2008—even though they are commonly implemented—since
 89727 there is no requirement that the FORTRAN compiler use the C preprocessor.

89728 The **-onetrip** option was not included in this volume of POSIX.1-2008, even though many
 89729 historical compilers support it, because it is derived from FORTRAN-66; it is an anachronism
 89730 that should not be perpetuated.

89731 Some implementations produce compilation listings. This aspect of FORTRAN has been left
 89732 unspecified because there was controversy concerning the various methods proposed for
 89733 implementing it: a **-V** option overlapped with historical vendor practice and a naming
 89734 convention of creating files with **.l** suffixes collided with historical *lex* file naming practice.

There is no **-I** option in this version of this volume of POSIX.1-2008 to specify a directory for file inclusion. An **INCLUDE** directive has been a part of the Fortran-90 discussions, but an interface supporting that standard is not in the current scope.

It is noted that many FORTRAN compilers produce an object module even when compilation errors occur; during a subsequent compilation, the compiler may patch the object module rather than recompiling all the code. Consequently, it is left to the implementor whether or not an object file is created.

A reference to MIL-STD-1753 was removed from an early proposal in response to a request from the POSIX FORTRAN-binding standard developers. It was not the intention of the standard developers to require certification of the FORTRAN compiler, and IEEE Std 1003.9-1992 does not specify the military standard or any special preprocessing requirements. Furthermore, use of that document would have been inappropriate for an international standard.

The specification of optimization has been subject to changes through early proposals. At one time, **-O** and **-N** were Booleans: optimize and do not optimize (with an unspecified default). Some historical practice led this to be changed to:

-O 0 No optimization.

-O 1 Some level of optimization.

-O n Other, unspecified levels of optimization.

It is not always clear whether “good code generation” is the same thing as optimization. Simple optimizations of local actions do not usually affect the semantics of a program. The **-O 0** option has been included to accommodate the very particular nature of scientific calculations in a highly optimized environment; compilers make errors. Some degree of optimization is expected, even if it is not documented here, and the ability to shut it off completely could be important when porting an application. An implementation may treat **-O 0** as “do less than normal” if it wishes, but this is only meaningful if any of the operations it performs can affect the semantics of a program. It is highly dependent on the implementation whether doing less than normal is logical. It is not the intent of the **-O 0** option to ask for inefficient code generation, but rather to assure that any semantically visible optimization is suppressed.

The specification of standard library access is consistent with the C compiler specification. Implementations are not required to have **/usr/lib/libf.a**, as many historical implementations do, but if not they are required to recognize **f** as a token.

External symbol size limits are in normative text; conforming applications need to know these limits. However, the minimum maximum symbol length should be taken as a constraint on a conforming application, not on an implementation, and consequently the action taken for a symbol exceeding the limit is unspecified. The minimum size for the external symbol table was added for similar reasons.

The CONSEQUENCES OF ERRORS section clearly specifies the behavior of the compiler when compilation or link-edit errors occur. The behavior of several historical implementations was examined, and the choice was made to be silent on the status of the executable, or **a.out**, file in the face of compiler or linker errors. If a linker writes the executable file, then links it on disk with **lseek()**s and **write()**s, the partially linked executable file can be left on disk and its execute bits turned off if the link edit fails. However, if the linker links the image in memory before writing the file to disk, it need not touch the executable file (if it already exists) because the link edit fails. Since both approaches are historical practice, a conforming application shall rely on the exit status of **fort77**, rather than on the existence or mode of the executable file.

The **-g** and **-s** options are not specified as mutually-exclusive. Historically, these two options

89781 have been mutually-exclusive, but because both are so loosely specified, it seemed appropriate
89782 to leave their interaction unspecified.

89783 The requirement that conforming applications specify compiler options separately is to reserve
89784 the multi-character option name space for vendor-specific compiler options, which are known to
89785 exist in many historical implementations. Implementations are not required to recognize, for
89786 example, `-gc` as if it were `-g -c`; nor are they forbidden from doing so. The SYNOPSIS shows all
89787 of the options separately to highlight this requirement on applications.

89788 Echoing filenames to standard error is considered a diagnostic message because it would
89789 otherwise be difficult to associate an error message with the erring file. They are described with
89790 “may” to allow implementations to use other methods of identifying files and to parallel the
89791 description in *c99*.

89792 FUTURE DIRECTIONS

89793 A compilation system based on the ISO/IEC 1539:1991 standard may be considered for a future
89794 version; it may have a different utility name from *fort77*.

89795 SEE ALSO

89796 *ar*, *asa*, *c99*, *umask*

89797 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

89798 XSH *exec*

89799 CHANGE HISTORY

89800 First released in Issue 4.

89801 Issue 6

89802 This utility is marked as part of the FORTRAN Development Utilities option.

89803 The normative text is reworded to avoid use of the term “must” for application requirements.

89804 Issue 7

89805 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

89806 **NAME**89807 *fuser* — list process IDs of all processes that have one or more files open89808 **SYNOPSIS**89809 XSI *fuser* [-cfu] *file...*89810 **DESCRIPTION**

89811 The *fuser* utility shall write to standard output the process IDs of processes running on the local
 89812 system that have one or more named files open. For block special devices, all processes using
 89813 any file on that device are listed.

89814 The *fuser* utility shall write to standard error additional information about the named files
 89815 indicating how the file is being used.

89816 Any output for processes running on remote systems that have a named file open is unspecified.

89817 A user may need appropriate privileges to invoke the *fuser* utility.

89818 **OPTIONS**

89819 The *fuser* utility shall conform to XBD Section 12.2 (on page 215).

89820 The following options shall be supported:

89821 **-c** The file is treated as a mount point and the utility shall report on any files open in
 89822 the file system.

89823 **-f** The report shall be only for the named files.

89824 **-u** The user name, in parentheses, associated with each process ID written to standard
 89825 output shall be written to standard error.

89826 **OPERANDS**

89827 The following operand shall be supported:

89828 *file* A pathname on which the file or file system is to be reported.

89829 **STDIN**

89830 Not used.

89831 **INPUT FILES**

89832 The user database.

89833 **ENVIRONMENT VARIABLES**

89834 The following environment variables shall affect the execution of *fuser*:

89835 **LANG** Provide a default value for the internationalization variables that are unset or null.
 89836 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 89837 variables used to determine the values of locale categories.)

89838 **LC_ALL** If set to a non-empty string value, override the values of all the other
 89839 internationalization variables.

89840 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 89841 characters (for example, single-byte as opposed to multi-byte characters in
 89842 arguments).

89843 **LC_MESSAGES**

89844 Determine the locale that should be used to affect the format and contents of
 89845 diagnostic messages written to standard error.

fuser*Utilities*

89846 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

89847 **ASYNCHRONOUS EVENTS**

89848 Default.

89849 **STDOUT**

89850 The *fuser* utility shall write the process ID for each process using each file given as an operand to
89851 standard output in the following format:

89852 "%d", <*process_id*>

89853 **STDERR**

89854 The *fuser* utility shall write diagnostic messages to standard error.

89855 The *fuser* utility also shall write the following to standard error:

- 89856 • The pathname of each named file is written followed immediately by a <colon>.
- 89857 • For each process ID written to standard output, the character 'c' shall be written to
89858 standard error if the process is using the file as its current directory and the character 'r'
89859 shall be written to standard error if the process is using the file as its root directory.
89860 Implementations may write other alphabetic characters to indicate other uses of files.
- 89861 • When the **-u** option is specified, characters indicating the use of the file shall be followed
89862 immediately by the user name, in parentheses, corresponding to the real user ID of the
89863 process. If the user name cannot be resolved from the real user ID of the process, the real
89864 user ID of the process shall be written instead of the user name.

89865 When standard output and standard error are directed to the same file, the output shall be
89866 interleaved so that the filename appears at the start of each line, followed by the process ID and
89867 characters indicating the use of the file. Then, if the **-u** option is specified, the user name or user
89868 ID for each process using that file shall be written.

89869 A <newline> shall be written to standard error after the last output described above for each *file*
89870 operand.

89871 **OUTPUT FILES**

89872 None.

89873 **EXTENDED DESCRIPTION**

89874 None.

89875 **EXIT STATUS**

89876 The following exit values shall be returned:

- 89877 0 Successful completion.
- 89878 >0 An error occurred.

89879 **CONSEQUENCES OF ERRORS**

89880 Default.

APPLICATION USAGE

None.

EXAMPLES

The command:

```
fuser -fu .
```

writes to standard output the process IDs of processes that are using the current directory and writes to standard error an indication of how those processes are using the directory and the user names associated with the processes that are using the current directory.

```
fuser -c <mount point>
```

writes to standard output the process IDs of processes that are using any file in the file system which is mounted on <mount point> and writes to standard error an indication of how those processes are using the files.

```
fuser <mount point>
```

writes to standard output the process IDs of processes that are using the file which is named by <mount point> and writes to standard error an indication of how those processes are using the file.

```
fuser <block device>
```

writes to standard output the process IDs of processes that are using any file which is on the device named by <block device> and writes to standard error an indication of how those processes are using the file.

```
fuser --f <block device>
```

writes to standard output the process IDs of processes that are using the file <block device> itself and writes to standard error an indication of how those processes are using the file.

RATIONALE

The definition of the *fuser* utility follows existing practice.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

CHANGE HISTORY

First released in Issue 5.

Issue 7

SD5-XCU-ERN-90 is applied, updating the EXAMPLES section.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

gencat

Utilities

89915 **NAME**89916 **gencat** — generate a formatted message catalog89917 **SYNOPSIS**89918 **gencat** *catfile* *msgfile*...89919 **DESCRIPTION**

89920 The *gencat* utility shall merge the message text source file *msgfile* into a formatted message
 89921 catalog *catfile*. The file *catfile* shall be created if it does not already exist. If *catfile* does exist, its
 89922 messages shall be included in the new *catfile*. If set and message numbers collide, the new
 89923 message text defined in *msgfile* shall replace the old message text currently contained in *catfile*.

89924 **OPTIONS**

89925 None.

89926 **OPERANDS**

89927 The following operands shall be supported:

89928 *catfile* A pathname of the formatted message catalog. If '-' is specified, standard output
 89929 shall be used. The format of the message catalog produced is unspecified.

89930 *msgfile* A pathname of a message text source file. If '-' is specified for an instance of
 89931 *msgfile*, standard input shall be used. The format of message text source files is
 89932 defined in the EXTENDED DESCRIPTION section.

89933 **STDIN**89934 The standard input shall not be used unless a *msgfile* operand is specified as '-'.89935 **INPUT FILES**

89936 The input files shall be text files.

89937 **ENVIRONMENT VARIABLES**89938 The following environment variables shall affect the execution of *gencat*:

89939 *LANG* Provide a default value for the internationalization variables that are unset or null.
 89940 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 89941 variables used to determine the values of locale categories.)

89942 *LC_ALL* If set to a non-empty string value, override the values of all the other
 89943 internationalization variables.

89944 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 89945 characters (for example, single-byte as opposed to multi-byte characters in
 89946 arguments and input files).

89947 *LC_MESSAGES*

89948 Determine the locale that should be used to affect the format and contents of
 89949 diagnostic messages written to standard error.

89950 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

89951 **ASYNCHRONOUS EVENTS**

89952 Default.

89953 **STDOUT**89954 The standard output shall not be used unless the *catfile* operand is specified as '-'.

89955 **STDERR**

89956 The standard error shall be used only for diagnostic messages.

89957 **OUTPUT FILES**

89958 None.

89959 **EXTENDED DESCRIPTION**

89960 The content of a message text file shall be in the format defined as follows. Note that the fields of
 89961 a message text source line are separated by a single <blank> character. Any other <blank>
 89962 characters are considered to be part of the subsequent field.

89963 **\$set** *n comment*

89964 This line specifies the set identifier of the following messages until the next **\$set** or
 89965 end-of-file appears. The *n* denotes the set identifier, which is defined as a number
 89966 in the range [1, {NL_SETMAX}] (see the <limits.h> header defined in the Base
 89967 Definitions volume of POSIX.1-2008). The application shall ensure that set
 89968 identifiers are presented in ascending order within a single source file, but need
 89969 not be contiguous. Any string following the set identifier shall be treated as a
 89970 comment. If no **\$set** directive is specified in a message text source file, all messages
 89971 shall be located in an implementation-defined default message set NL_SETD (see
 89972 the <nl_types.h> header defined in the Base Definitions volume of POSIX.1-2008).

89973 **\$delset** *n comment*

89974 This line deletes message set *n* from an existing message catalog. The *n* denotes the
 89975 set number [1, {NL_SETMAX}]. Any string following the set number shall be
 89976 treated as a comment.

89977 **\$ comment** A line beginning with ' \$ ' followed by a <blank> shall be treated as a comment.

89978 *m message-text*

89979 The *m* denotes the message identifier, which is defined as a number in the range [1,
 89980 {NL_MSGMAX}] (see the <limits.h> header). The *message-text* shall be stored in the
 89981 message catalog with the set identifier specified by the last **\$set** directive, and with
 89982 message identifier *m*. If the *message-text* is empty, and a <blank> field separator is
 89983 present, an empty string shall be stored in the message catalog. If a message source
 89984 line has a message number, but neither a field separator nor *message-text*, the
 89985 existing message with that number (if any) shall be deleted from the catalog. The
 89986 application shall ensure that message identifiers are in ascending order within a
 89987 single set, but need not be contiguous. The application shall ensure that the length
 89988 of *message-text* is in the range [0, {NL_TEXTMAX}] (see the <limits.h> header).

89989 **\$quote** *n* This line specifies an optional quote character *c*, which can be used to surround
 89990 *message-text* so that trailing <space> characters or null (empty) messages are visible
 89991 in a message source line. By default, or if an empty **\$quote** directive is supplied, no
 89992 quoting of *message-text* shall be recognized.

89993 Empty lines in a message text source file shall be ignored. The effects of lines starting with any
 89994 character other than those defined above are implementation-defined.

89995 Text strings can contain the special characters and escape sequences defined in the following
 89996 table:

89997
89998
89999
90000
90001
90002
90003
90004
90005

90006
90007
90008

90009
90010

90011
90012

90013

90014

90015
90016
90017
90018

90019
90020

90021
90022
90023
90024

90025
90026

90027
90028

90029
90030

90031
90032

90033

90034
90035

90036
90037

Description	Symbol	Sequence
<newline>	NL(LF)	\n
Horizontal-tab	HT	\t
<vertical-tab>	VT	\v
<backspace>	BS	\b
<carriage-return>	CR	\r
<form-feed>	FF	\f
Backslash	\	\\
Bit pattern	ddd	\ddd

The escape sequence "\ddd" consists of <backslash> followed by one, two, or three octal digits, which shall be taken to specify the value of the desired character. If the character following a <backslash> is not one of those specified, the <backslash> shall be ignored.

A <backslash> followed by a <newline> is also used to continue a string on the following line. Thus, the following two lines describe a single message string:

1 This line continues \
to the next line

which shall be equivalent to:

1 This line continues to the next line

EXIT STATUS

The following exit values shall be returned:

- 0 Successful completion.
- >0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Message catalogs produced by *gencat* are binary encoded, meaning that their portability cannot be guaranteed between different types of machine. Thus, just as C programs need to be recompiled for each type of machine, so message catalogs must be recreated via *gencat*.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iconv

XBD Chapter 8 (on page 173), <limits.h>, <nl_types.h>

CHANGE HISTORY

First released in Issue 3.

Issue 6

The normative text is reworded to avoid use of the term “must” for application requirements.

*Utilities***gencat**

90038 **Issue 7**
90039

The *gencat* utility is moved from the XSI option to the Base.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

get*Utilities*90040 **NAME**90041 `get` — get a version of an SCCS file (**DEVELOPMENT**)90042 **SYNOPSIS**90043 XSI `get [-begkmnlLpst] [-c cutoff] [-i list] [-r SID] [-x list] file...`90044 **DESCRIPTION**90045 The `get` utility shall generate a text file from each named SCCS *file* according to the specifications
90046 given by its options.90047 The generated text shall normally be written into a file called the **g-file** whose name is derived
90048 from the SCCS filename by simply removing the leading "s.". 90049 **OPTIONS**90050 The `get` utility shall conform to XBD [Section 12.2](#) (on page 215).

90051 The following options shall be supported:

90052 **-r *SID*** Indicate the SCCS Identification String (SID) of the version (delta) of an SCCS file
90053 to be retrieved. The table shows, for the most useful cases, what version of an
90054 SCCS file is retrieved (as well as the SID of the version to be eventually created by
90055 *delta* if the **-e** option is also used), as a function of the SID specified.90056 **-c *cutoff*** Indicate the *cutoff* date-time, in the form:90057 `YY[MM[DD[HH[MM[SS]]]]]`90058 For the YY component, values in the range [69,99] shall refer to years 1969 to 1999
90059 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.90060 **Note:** It is expected that in a future version of this standard the default century inferred
90061 from a 2-digit year will change. (This would apply to all commands accepting a
90062 2-digit year as input.)90063 No changes (deltas) to the SCCS file that were created after the specified *cutoff*
90064 date-time shall be included in the generated text file. Units omitted from the date-
90065 time default to their maximum possible values; for example, **-c 7502** is equivalent
90066 to **-c 750228235959**.90067 Any number of non-numeric characters may separate the various 2-digit pieces of
90068 the *cutoff* date-time. This feature allows the user to specify a *cutoff* date in the form:
90069 **-c "77/2/2 9:22:25"**.90070 **-e** Indicate that the `get` is for the purpose of editing or making a change (delta) to the
90071 SCCS file via a subsequent use of *delta*. The **-e** option used in a `get` for a particular
90072 version (SID) of the SCCS file shall prevent further `get` commands from editing on
90073 the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file.
90074 Concurrent use of `get -e` for different SIDs is always allowed.90075 If the **g-file** generated by `get` with a **-e** option is accidentally ruined in the process
90076 of editing, it may be regenerated by re-executing the `get` command with the **-k**
90077 option in place of the **-e** option.90078 SCCS file protection specified via the ceiling, floor, and authorized user list stored
90079 in the SCCS file shall be enforced when the **-e** option is used.90080 **-b** Use with the **-e** option to indicate that the new delta should have an SID in a new
90081 branch as shown in the table below. This option shall be ignored if the **b** flag is not
90082 present in the file or if the retrieved delta is not a leaf delta. (A leaf delta is one that
90083 has no successors on the SCCS file tree.)

Utilities

get

90084		Note: A branch delta may always be created from a non-leaf delta.
90085	-i list	Indicate a <i>list</i> of deltas to be included (forced to be applied) in the creation of the generated file. The <i>list</i> has the following syntax:
90086		
90087		<code><list> ::= <range> <list> , <range></code>
90088		<code><range> ::= SID SID - SID</code>
90089		SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of the table in the EXTENDED DESCRIPTION section, except
90090		that the result of supplying a partial SID is unspecified. A diagnostic message shall
90091		be written if the first SID in the range is not an ancestor of the second SID in the
90092		range.
90093		
90094	-x list	Indicate a <i>list</i> of deltas to be excluded (forced not to be applied) in the creation of
90095		the generated file. See the -i option for the <i>list</i> format.
90096	-k	Suppress replacement of identification keywords (see below) in the retrieved text
90097		by their value. The -k option shall be implied by the -e option.
90098	-l	Write a delta summary into an l-file .
90099	-L	Write a delta summary to standard output. All informative output that normally is
90100		written to standard output shall be written to standard error instead, unless the -s
90101		option is used, in which case it shall be suppressed.
90102	-p	Write the text retrieved from the SCCS file to the standard output. No g-file shall
90103		be created. All informative output that normally goes to the standard output shall
90104		go to standard error instead, unless the -s option is used, in which case it shall
90105		disappear.
90106	-s	Suppress all informative output normally written to standard output. However,
90107		fatal error messages (which shall always be written to the standard error) shall
90108		remain unaffected.
90109	-m	Precede each text line retrieved from the SCCS file by the SID of the delta that
90110		inserted the text line in the SCCS file. The format shall be:
90111		<code>"%s\t%s", <SID>, <text line></code>
90112	-n	Precede each generated text line with the %M% identification keyword value (see
90113		below). The format shall be:
90114		<code>"%s\t%s", <%M% value>, <text line></code>
90115		When both the -m and -n options are used, the <code><text line></code> shall be replaced by the
90116		-m option-generated format.
90117	-g	Suppress the actual retrieval of text from the SCCS file. It is primarily used to
90118		generate an l-file , or to verify the existence of a particular SID.
90119	-t	Use to access the most recently created (top) delta in a given release (for example,
90120		-r 1), or release and level (for example, -r 1.2).

get*Utilities*90121 **OPERANDS**

90122 The following operands shall be supported:

90123 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *get*
 90124 utility shall behave as though each file in the directory were specified as a named
 90125 file, except that non-SCCS files (last component of the pathname does not begin
 90126 with **s**.) and unreadable files shall be silently ignored.

90127 If exactly one *file* operand appears, and it is **'-'**, the standard input shall be read;
 90128 each line of the standard input is taken to be the name of an SCCS file to be
 90129 processed. Non-SCCS files and unreadable files shall be silently ignored.

90130 **STDIN**

90131 The standard input shall be a text file used only if the *file* operand is specified as **'-'**. Each line
 90132 of the text file shall be interpreted as an SCCS pathname.

90133 **INPUT FILES**

90134 The SCCS files shall be files of an unspecified format.

90135 **ENVIRONMENT VARIABLES**90136 The following environment variables shall affect the execution of *get*:

90137 *LANG* Provide a default value for the internationalization variables that are unset or null.
 90138 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 90139 variables used to determine the values of locale categories.)

90140 *LC_ALL* If set to a non-empty string value, override the values of all the other
 90141 internationalization variables.

90142 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 90143 characters (for example, single-byte as opposed to multi-byte characters in
 90144 arguments and input files).

90145 *LC_MESSAGES*

90146 Determine the locale that should be used to affect the format and contents of
 90147 diagnostic messages written to standard error, and informative messages written
 90148 to standard output (or standard error, if the **-p** option is used).

90149 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

90150 *TZ* Determine the timezone in which the times and dates written in the SCCS file are
 90151 evaluated. If the *TZ* variable is unset or NULL, an unspecified system default
 90152 timezone is used.

90153 **ASYNCHRONOUS EVENTS**

90154 Default.

90155 **STDOUT**

90156 For each file processed, *get* shall write to standard output the SID being accessed and the
 90157 number of lines retrieved from the SCCS file, in the following format:

90158 "%s\n%d lines\n", <SID>, <number of lines>

90159 If the **-e** option is used, the SID of the delta to be made shall appear after the SID accessed and
 90160 before the number of lines generated, in the POSIX locale:

90161 "%s\nnew delta %s\n%d lines\n", <SID accessed>,
 90162 <SID to be made>, <number of lines>

90163 If there is more than one named file or if a directory or standard input is named, each pathname

shall be written before each of the lines shown in one of the preceding formats:

```
"\n%s:\n", <pathname>
```

If the **-L** option is used, a delta summary shall be written following the format specified below for **l-files**.

If the **-i** option is used, included deltas shall be listed following the notation, in the POSIX locale:

```
"Included:\n"
```

If the **-x** option is used, excluded deltas shall be listed following the notation, in the POSIX locale:

```
"Excluded:\n"
```

If the **-p** or **-L** options are specified, the standard output shall consist of the text retrieved from the SCCS file.

STDERR

The standard error shall be used only for diagnostic messages, except if the **-p** or **-L** options are specified, it shall include all informative messages normally sent to standard output.

OUTPUT FILES

Several auxiliary files may be created by *get*. These files are known generically as the **g-file**, **l-file**, **p-file**, and **z-file**. The letter before the <hyphen> is called the *tag*. An auxiliary filename shall be formed from the SCCS filename: the application shall ensure that the last component of all SCCS filenames is of the form *s.module-name*; the auxiliary files shall be named by replacing the leading *s* with the tag. The **g-file** shall be an exception to this scheme: the **g-file** is named by removing the *s*. prefix. For example, for *s.xyz.c*, the auxiliary filenames would be *xyz.c*, *l.xyz.c*, *p.xyz.c*, and *z.xyz.c*, respectively.

The **g-file**, which contains the generated text, shall be created in the current directory (unless the **-p** option is used). A **g-file** shall be created in all cases, whether or not any lines of text were generated by the *get*. It shall be owned by the real user. If the **-k** option is used or implied, the **g-file** shall be writable by the owner only (read-only for everyone else); otherwise, it shall be read-only. Only the real user need have write permission in the current directory.

The **l-file** shall contain a table showing which deltas were applied in generating the retrieved text. The **l-file** shall be created in the current directory if the **-l** option is used; it shall be read-only and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the **l-file** shall have the following format:

```
"%c%c%cΔ%s\t%sΔ%s\n", <code1>, <code2>, <code3>,
<SID>, <date-time>, <login>
```

where the entries are:

<code1> A <space> if the delta was applied; ' * ' otherwise.

<code2> A <space> if the delta was applied or was not applied and ignored; ' * ' if the delta was not applied and was not ignored.

<code3> A character indicating a special reason why the delta was or was not applied:

I Included.

get*Utilities*

- 90205 **X** Excluded.
- 90206 **C** Cut off (by a **-c** option).
- 90207 <*date-time*> Date and time (using the format of the *date* utility's %Y/%m/%d %T conversion specification format) of creation.
- 90208
- 90209 <*login*> Login name of person who created *delta*.
- 90210 The comments and MR data shall follow on subsequent lines, indented one <tab>. A blank line shall terminate each entry.
- 90211
- 90212 The **p-file** shall be used to pass information resulting from a *get* with a **-e** option along to *delta*. Its contents shall also be used to prevent a subsequent execution of *get* with a **-e** option for the same SID until *delta* is executed or the joint edit flag, **j**, is set in the SCCS file. The **p-file** shall be created in the directory containing the SCCS file and the application shall ensure that the effective user has write permission in that directory. It shall be writable by owner only, and owned by the effective user. Each line in the **p-file** shall have the following format:
- 90213
- 90214
- 90215
- 90216
- 90217
- 90218 "%sΔ%sΔ%sΔ%s%s\n", <*g-file SID*>,
 90219 <*SID of new delta*>, <*login-name of real user*>,
 90220 <*date-time*>, <*i-value*>, <*x-value*>
- 90221 where <*i-value*> uses the format " " if no **-i** option was specified, and shall use the format:
- 90222 "Δ-*i*%s", <**-i** option *option-argument*>
- 90223 if a **-i** option was specified and <*x-value*> uses the format " " if no **-x** option was specified, and shall use the format:
- 90224
- 90225 "Δ-*x*%s", <**-x** option *option-argument*>
- 90226 if a **-x** option was specified. There can be an arbitrary number of lines in the **p-file** at any time; no two lines shall have the same new delta SID.
- 90227
- 90228 The **z-file** shall serve as a lock-out mechanism against simultaneous updates. Its contents shall be the binary process ID of the command (that is, *get*) that created it. The **z-file** shall be created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the **p-file** shall apply for the **z-file**. The **z-file** shall be created read-only.
- 90229
- 90230
- 90231

90232 EXTENDED DESCRIPTION

Determination of SCCS Identification String				
SID* Specified	–b Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	–	R < mR and R does not exist	hR.mL**	hR.mL.(mB+1).1
R	–	Trunk successor in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk successor	R.L	R.(L+1)
R.L	yes	No trunk successor	R.L	R.L.(mB+1).1
R.L	–	Trunk successor in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch successor	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch successor	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch successor	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch successor	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch successor	R.L.B.S	R.L.(mB+1).1

* R, L, B, and S are the release, level, branch, and sequence components of the SID, respectively; m means maximum. Thus, for example, R.mL means “the maximum level number within release R”; R.L.(mB+1).1 means “the first sequence number on the new branch (that is, maximum branch number plus one) of level L within release R”. Note that if the SID specified is of the form R.L, R.L.B, or R.L.B.S, each of the specified components shall exist.

** hR is the highest existing release that is lower than the specified, nonexistent, release R.

*** This is used to force creation of the first delta in a new release.

† The –b option is effective only if the b flag is present in the file. An entry of ‘–’ means “irrelevant”.

‡ This case applies if the d (default SID) flag is not present in the file. If the d flag is present in the file, then the SID obtained from the d flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

90268 **System Date and Time**

90269 When a **g-file** is generated, the creation time of deltas in the SCCS file may be taken into
 90270 account. If any of these times are apparently in the future, the behavior is unspecified.

90271 **Identification Keywords**

90272 Identifying information shall be inserted into the text retrieved from the SCCS file by replacing
 90273 identification keywords with their value wherever they occur. The following keywords may be
 90274 used in the text stored in an SCCS file:

90275	%M%	Module name: either the value of the m flag in the file, or if absent, the name of the
90276		SCCS file with the leading s. removed.
90277	%I%	SCCS identification (SID) (%R%.%L% or %R%.%L%.%B%.%S%) of the retrieved
90278		text.
90279	%R%	Release.
90280	%L%	Level.
90281	%B%	Branch.
90282	%S%	Sequence.
90283	%D%	Current date (YY/MM/DD).
90284	%H%	Current date (MM/DD/YY).
90285	%T%	Current time (HH:MM:SS).
90286	%E%	Date newest applied delta was created (YY/MM/DD).
90287	%G%	Date newest applied delta was created (MM/DD/YY).
90288	%U%	Time newest applied delta was created (HH:MM:SS).
90289	%Y%	Module type: value of the t flag in the SCCS file.
90290	%F%	SCCS filename.
90291	%P%	SCCS absolute pathname.
90292	%Q%	The value of the q flag in the file.
90293	%C%	Current line number. This keyword is intended for identifying messages output by
90294		the program, such as "this should not have happened" type errors. It is not
90295		intended to be used on every line to provide sequence numbers.
90296	%Z%	The four-character string "@ (#)" recognizable by <i>what</i> .
90297	%W%	A shorthand notation for constructing <i>what</i> strings:
90298		%W%=%Z%%M%<tab>%I%
90299	%A%	Another shorthand notation for constructing <i>what</i> strings:
90300		%A%=%Z%%Y%%M%%I%%Z%

90301 **EXIT STATUS**

90302 The following exit values shall be returned:

90303 0 Successful completion.

90304 >0 An error occurred.

90305 CONSEQUENCES OF ERRORS

90306 Default.

90307 APPLICATION USAGE

90308 Problems can arise if the system date and time have been modified (for example, put forward
90309 and then back again, or unsynchronized clocks across a network) and can also arise when
90310 different values of the *TZ* environment variable are used.

90311 Problems of a similar nature can also arise for the operation of the *delta* utility, which compares
90312 the previous file body against the working file as part of its normal operation.

90313 EXAMPLES

90314 None.

90315 RATIONALE

90316 None.

90317 FUTURE DIRECTIONS

90318 None.

90319 SEE ALSO

90320 *admin*, *delta*, *prs*, *what*

90321 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

90322 CHANGE HISTORY

90323 First released in Issue 2.

90324 Issue 5

90325 A correction is made to the first format string in STDOUT.

90326 The interpretation of the *YY* component of the *-c cutoff* argument is noted.

90327 Issue 6

90328 The obsolescent SYNOPSIS is removed, removing the *-lp* option.

90329 The normative text is reworded to avoid use of the term “must” for application requirements.

90330 The Open Group Corrigendum U025/5 is applied, correcting text in the OPTIONS section.

90331 The Open Group Corrigendum U048/1 is applied.

90332 The Open Group Interpretation PIN4C.00014 is applied.

90333 The Open Group Base Resolution bwg2001-007 is applied as follows:

90334 • The EXTENDED DESCRIPTION section is updated to make partial SID handling
90335 unspecified, reflecting common usage, and to clarify SID ranges.

90336 • New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections
90337 regarding how the system date and time may be taken into account.

90338 • The *TZ* environment variable is added to the ENVIRONMENT VARIABLES section.

90339 Issue 7

90340 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

getconf

Utilities

90341 **NAME**90342 `getconf` — get configuration values90343 **SYNOPSIS**90344 `getconf [-v specification] system_var`90345 `getconf [-v specification] path_var pathname`90346 **DESCRIPTION**90347 In the first synopsis form, the *getconf* utility shall write to the standard output the value of the
90348 variable specified by the *system_var* operand.90349 In the second synopsis form, the *getconf* utility shall write to the standard output the value of the
90350 variable specified by the *path_var* operand for the path specified by the *pathname* operand.90351 The value of each configuration variable shall be determined as if it were obtained by calling the
90352 function from which it is defined to be available by this volume of POSIX.1-2008 or by the
90353 System Interfaces volume of POSIX.1-2008 (see the OPERANDS section). The value shall reflect
90354 conditions in the current operating environment.90355 **OPTIONS**90356 The *getconf* utility shall conform to XBD Section 12.2 (on page 215).

90357 The following option shall be supported:

90358 **-v specification**90359 Indicate a specific specification and version for which configuration variables shall
90360 be determined. If this option is not specified, the values returned correspond to an
90361 implementation default conforming compilation environment.

90362 If the command:

90363 `getconf _POSIX_V7_ILP32_OFF32`90364 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of
90365 the form:90366 `getconf -v _POSIX_V7_ILP32_OFF32 ...`90367 determine values for configuration variables corresponding to the
90368 POSIX_V7_ILP32_OFF32 compilation environment specified in *c99*, the
90369 EXTENDED DESCRIPTION.

90370 If the command:

90371 `getconf _POSIX_V7_ILP32_OFFBIG`90372 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of
90373 the form:90374 `getconf -v POSIX_V7_ILP32_OFFBIG ...`90375 determine values for configuration variables corresponding to the
90376 POSIX_V7_ILP32_OFFBIG compilation environment specified in *c99*, the
90377 EXTENDED DESCRIPTION.

90378 If the command:

90379 `getconf _POSIX_V7_LP64_OFF64`90380 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of
90381 the form:

Utilities

getconf

90382 `getconf -v POSIX_V7_LP64_OFF64 ...`
 90383 determine values for configuration variables corresponding to the
 90384 POSIX_V7_LP64_OFF64 compilation environment specified in *c99*, the
 90385 EXTENDED DESCRIPTION.

90386 If the command:

90387 `getconf _POSIX_V7_LPBIG_OFFBIG`

90388 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of
 90389 the form:

90390 `getconf -v POSIX_V7_LPBIG_OFFBIG ...`

90391 determine values for configuration variables corresponding to the
 90392 POSIX_V7_LPBIG_OFFBIG compilation environment specified in *c99*, the
 90393 EXTENDED DESCRIPTION.

90394 OPERANDS

90395 The following operands shall be supported:

90396 *path_var* A name of a configuration variable. All of the variables in the Variable column of
 90397 the table in the DESCRIPTION of the *fpathconf()* function defined in the System
 90398 Interfaces volume of POSIX.1-2008, without the enclosing braces, shall be
 90399 supported. The implementation may add other local variables.

90400 *pathname* A pathname for which the variable specified by *path_var* is to be determined.

90401 *system_var* A name of a configuration variable. All of the following variables shall be
 90402 supported:

- 90403 • The names in the Variable column of the table in the DESCRIPTION of the
- 90404 *sysconf()* function in the System Interfaces volume of POSIX.1-2008, except
- 90405 for the entries corresponding to `_SC_CLK_TCK`, `_SC_GETGR_R_SIZE_MAX`,
- 90406 and `_SC_GETPW_R_SIZE_MAX`, without the enclosing braces.

90407 For compatibility with earlier versions, the following variable names shall
 90408 also be supported:

90409 `POSIX2_C_BIND`
 90410 `POSIX2_C_DEV`
 90411 `POSIX2_CHAR_TERM`
 90412 `POSIX2_FORT_DEV`
 90413 `POSIX2_FORT_RUN`
 90414 `POSIX2_LOCALEDEF`
 90415 `POSIX2_SW_DEV`
 90416 `POSIX2_UPE`
 90417 `POSIX2_VERSION`

90418 and shall be equivalent to the same name prefixed with an <underscore>.
 90419 This requirement may be removed in a future version.

- 90420 • The names of the symbolic constants used as the *name* argument of the
- 90421 *confstr()* function in the System Interfaces volume of POSIX.1-2008, without
- 90422 the `_CS_` prefix.

getconf*Utilities*

90423		• The names of the symbolic constants listed under the headings “Maximum Values” and “Minimum Values” in the description of the <limits.h> header in the Base Definitions volume of POSIX.1-2008, without the enclosing braces.
90424		
90425		
90426		
90427		For compatibility with earlier versions, the following variable names shall also be supported:
90428		
90429		POSIX2_BC_BASE_MAX
90430		POSIX2_BC_DIM_MAX
90431		POSIX2_BC_SCALE_MAX
90432		POSIX2_BC_STRING_MAX
90433		POSIX2_COLL_WEIGHTS_MAX
90434		POSIX2_EXPR_NEST_MAX
90435		POSIX2_LINE_MAX
90436		POSIX2_RE_DUP_MAX
90437		and shall be equivalent to the same name prefixed with an <underscore>.
90438		This requirement may be removed in a future version.
90439		The implementation may add other local values.
90440	STDIN	
90441		Not used.
90442	INPUT FILES	
90443		None.
90444	ENVIRONMENT VARIABLES	
90445		The following environment variables shall affect the execution of <i>getconf</i> :
90446	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
90447		
90448		
90449	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
90450		
90451	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
90452		
90453		
90454	<i>LC_MESSAGES</i>	
90455		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
90456		
90457	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
90458	ASYNCHRONOUS EVENTS	
90459		Default.
90460	STDOUT	
90461		If the specified variable is defined on the system and its value is described to be available from the <i>confstr()</i> function defined in the System Interfaces volume of POSIX.1-2008, its value shall be written in the following format:
90462		
90463		
90464		"%s\n", <value>
90465		Otherwise, if the specified variable is defined on the system, its value shall be written in the

90466 following format:

90467 "%d\n", <value>

90468 If the specified variable is valid, but is undefined on the system, *getconf* shall write using the
90469 following format:

90470 "undefined\n"

90471 If the variable name is invalid or an error occurs, nothing shall be written to standard output.

90472 **STDERR**

90473 The standard error shall be used only for diagnostic messages.

90474 **OUTPUT FILES**

90475 None.

90476 **EXTENDED DESCRIPTION**

90477 None.

90478 **EXIT STATUS**

90479 The following exit values shall be returned:

90480 0 The specified variable is valid and information about its current state was written
90481 successfully.

90482 >0 An error occurred.

90483 **CONSEQUENCES OF ERRORS**

90484 Default.

90485 **APPLICATION USAGE**

90486 None.

90487 **EXAMPLES**

90488 The following example illustrates the value of {NGROUPS_MAX}:

90489 `getconf NGROUPS_MAX`

90490 The following example illustrates the value of {NAME_MAX} for a specific directory:

90491 `getconf NAME_MAX /usr`

90492 The following example shows how to deal more carefully with results that might be unspecified:

```
90493 if value=$(getconf PATH_MAX /usr); then
90494     if [ "$value" = "undefined" ]; then
90495         echo PATH_MAX in /usr is indeterminate.
90496     else
90497         echo PATH_MAX in /usr is $value.
90498     fi
90499 else
90500     echo Error in getconf.
90501 fi
```

90502 Note that:

90503 `sysconf (_SC_2_C_BIND) ;`

90504 and:

90505 `system("getconf _POSIX2_C_BIND") ;`

getconf*Utilities*

in a C program could give different answers. The *sysconf()* call supplies a value that corresponds to the conditions when the program was either compiled or executed, depending on the implementation; the *system()* call to *getconf* always supplies a value corresponding to conditions when the program is executed.

RATIONALE

The original need for this utility, and for the *confstr()* function, was to provide a way of finding the configuration-defined default value for the *PATH* environment variable. Since *PATH* can be modified by the user to include directories that could contain utilities replacing the standard utilities, shell scripts need a way to determine the system-supplied *PATH* environment variable value that contains the correct search path for the standard utilities. It was later suggested that access to the other variables described in this volume of POSIX.1-2008 could also be useful to applications.

This functionality of *getconf* would not be adequately subsumed by another command such as:

```
grep var /etc/conf
```

because such a strategy would provide correct values for neither those variables that can vary at runtime, nor those that can vary depending on the path.

Early proposal versions of *getconf* specified exit status 1 when the specified variable was valid, but not defined on the system. The output string "undefined" is now used to specify this case with exit code 0 because so many things depend on an exit code of zero when an invoked utility is successful.

FUTURE DIRECTIONS

None.

SEE ALSO

c99

XBD Chapter 8 (on page 173), Section 12.2 (on page 215), *<limits.h>*

XSH *confstr()*, *fpathconf()*, *sysconf()*, *system()*

CHANGE HISTORY

First released in Issue 4.

Issue 5

In the OPERANDS section:

- {NL_MAX} is changed to {NL_NMAX}.
- Entries beginning NL_ are deleted from the list of standard configuration variables.
- The list of variables previously marked UX is merged with the list marked EX.
- Operands are added to support new Option Groups.
- Operands are added so that *getconf* can determine supported programming environments.

Issue 6

The Open Group Corrigendum U029/4 is applied, correcting the example command in the last paragraph of the OPTIONS section.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 90546 • Operands are added to determine supported programming environments.
- 90547 This reference page is updated for alignment with the ISO/IEC 9899:1999 standard. Specifically,
90548 new macros for *c99* programming environments are introduced.
- 90549 XSI marked *system_var* (XBS5_*) values are marked LEGACY.
- 90550 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/27 is applied, correcting the descriptions
90551 of *path_var* and *system_var* in the OPERANDS section.
- 90552 **Issue 7**
- 90553 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 90554 The EXAMPLES section is corrected.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

getopts*Utilities*90555 **NAME**90556 `getopts` — parse utility options90557 **SYNOPSIS**90558 `getopts optstring name [arg...]`90559 **DESCRIPTION**

90560 The *getopts* utility shall retrieve options and option-arguments from a list of parameters. It shall
 90561 support the Utility Syntax Guidelines 3 to 10, inclusive, described in XBD [Section 12.2](#) (on page
 90562 215).

90563 Each time it is invoked, the *getopts* utility shall place the value of the next option in the shell
 90564 variable specified by the *name* operand and the index of the next argument to be processed in the
 90565 shell variable *OPTIND*. Whenever the shell is invoked, *OPTIND* shall be initialized to 1.

90566 When the option requires an option-argument, the *getopts* utility shall place it in the shell
 90567 variable *OPTARG*. If no option was found, or if the option that was found does not have an
 90568 option-argument, *OPTARG* shall be unset.

90569 If an option character not contained in the *optstring* operand is found where an option character
 90570 is expected, the shell variable specified by *name* shall be set to the <question-mark> ('?')
 90571 character. In this case, if the first character in *optstring* is a <colon> (':'), the shell variable
 90572 *OPTARG* shall be set to the option character found, but no output shall be written to standard
 90573 error; otherwise, the shell variable *OPTARG* shall be unset and a diagnostic message shall be
 90574 written to standard error. This condition shall be considered to be an error detected in the way
 90575 arguments were presented to the invoking application, but shall not be an error in *getopts*
 90576 processing.

90577 If an option-argument is missing:

- 90578 • If the first character of *optstring* is a <colon>, the shell variable specified by *name* shall be
 90579 set to the <colon> character and the shell variable *OPTARG* shall be set to the option
 90580 character found.
- 90581 • Otherwise, the shell variable specified by *name* shall be set to the <question-mark>
 90582 character, the shell variable *OPTARG* shall be unset, and a diagnostic message shall be
 90583 written to standard error. This condition shall be considered to be an error detected in the
 90584 way arguments were presented to the invoking application, but shall not be an error in
 90585 *getopts* processing; a diagnostic message shall be written as stated, but the exit status shall
 90586 be zero.

90587 When the end of options is encountered, the *getopts* utility shall exit with a return value greater
 90588 than zero; the shell variable *OPTIND* shall be set to the index of the first non-option-argument,
 90589 where the first "--" argument is considered to be an option-argument if there are no other non-
 90590 option-arguments appearing before it, or the value "\$#+1" if there are no non-option-
 90591 arguments; the *name* variable shall be set to the <question-mark> character. Any of the following
 90592 shall identify the end of options: the special option "--", finding an argument that does not
 90593 begin with a '-', or encountering an error.

90594 The shell variables *OPTIND* and *OPTARG* shall be local to the caller of *getopts* and shall not be
 90595 exported by default.

90596 The shell variable specified by the *name* operand, *OPTIND*, and *OPTARG* shall affect the current
 90597 shell execution environment; see [Section 2.12](#) (on page 2331).

90598 If the application sets *OPTIND* to the value 1, a new set of parameters can be used: either the
 90599 current positional parameters or new *arg* values. Any other attempt to invoke *getopts* multiple
 90600 times in a single shell execution environment with parameters (positional parameters or *arg*

Utilities

getopts

operands) that are not the same in all invocations, or with an *OPTIND* value modified to be a value other than 1, produces unspecified results.

OPTIONS

None.

OPERANDS

The following operands shall be supported:

optstring A string containing the option characters recognized by the utility invoking *getopts*. If a character is followed by a <colon>, the option shall be expected to have an argument, which should be supplied as a separate argument. Applications should specify an option character and its option-argument as separate arguments, but *getopts* shall interpret the characters following an option character requiring arguments as an argument whether or not this is done. An explicit null option-argument need not be recognized if it is not supplied as a separate argument when *getopts* is invoked. (See also the *getopt()* function defined in the System Interfaces volume of POSIX.1-2008.) The characters <question-mark> and <colon> shall not be used as option characters by an application. The use of other option characters that are not alphanumeric produces unspecified results. If the option-argument is not supplied as a separate argument from the option character, the value in *OPTARG* shall be stripped of the option character and the ' - '. The first character in *optstring* determines how *getopts* behaves if an option character is not known or an option-argument is missing.

name The name of a shell variable that shall be set by the *getopts* utility to the option character that was found.

The *getopts* utility by default shall parse positional parameters passed to the invoking shell procedure. If *args* are given, they shall be parsed instead of the positional parameters.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *getopts*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

getopts*Utilities*

90644 *OPTIND* This variable shall be used by the *getopts* utility as the index of the next argument
 90645 to be processed.

90646 **ASYNCHRONOUS EVENTS**

90647 Default.

90648 **STDOUT**

90649 Not used.

90650 **STDERR**

90651 Whenever an error is detected and the first character in the *optstring* operand is not a <colon>
 90652 (' : '), a diagnostic message shall be written to standard error with the following information in
 90653 an unspecified format:

- 90654 • The invoking program name shall be identified in the message. The invoking program
 90655 name shall be the value of the shell special parameter 0 (see [Section 2.5.2](#), on page 2302) at
 90656 the time the *getopts* utility is invoked. A name equivalent to:

90657 `basename "$0"`

90658 may be used.

- 90659 • If an option is found that was not specified in *optstring*, this error is identified and the
 90660 invalid option character shall be identified in the message.

- 90661 • If an option requiring an option-argument is found, but an option-argument is not found,
 90662 this error shall be identified and the invalid option character shall be identified in the
 90663 message.

90664 **OUTPUT FILES**

90665 None.

90666 **EXTENDED DESCRIPTION**

90667 None.

90668 **EXIT STATUS**

90669 The following exit values shall be returned:

90670 0 An option, specified or unspecified by *optstring*, was found.

90671 >0 The end of options was encountered or an error occurred.

90672 **CONSEQUENCES OF ERRORS**

90673 Default.

90674 **APPLICATION USAGE**

90675 Since *getopts* affects the current shell execution environment, it is generally provided as a shell
 90676 regular built-in. If it is called in a subshell or separate utility execution environment, such as one
 90677 of the following:

90678 `(getopts abc value "$@")`

90679 `nohup getopts ...`

90680 `find . -exec getopts ... \;`

90681 it does not affect the shell variables in the caller's environment.

90682 Note that shell functions share *OPTIND* with the calling shell even though the positional
 90683 parameters are changed. If the calling shell and any of its functions uses *getopts* to parse
 90684 arguments, the results are unspecified.

EXAMPLES

The following example script parses and displays its arguments:

```
aflag=
bflag=
while getopts ab: name
do
    case $name in
        a)    aflag=1;;
        b)    bflag=1
              bval="$OPTARG";;
        ?)    printf "Usage: %s: [-a] [-b value] args\n" $0
              exit 2;;
    esac
done
if [ ! -z "$aflag" ]; then
    printf "Option -a specified\n"
fi
if [ ! -z "$bflag" ]; then
    printf 'Option -b "%s" specified\n' "$bval"
fi
shift $(( $OPTIND - 1 ))
printf "Remaining arguments are: %s\n" "$@"
```

RATIONALE

The *getopts* utility was chosen in preference to the System V *getopt* utility because *getopts* handles option-arguments containing <blank> characters.

The *OPTARG* variable is not mentioned in the ENVIRONMENT VARIABLES section because it does not affect the execution of *getopts*; it is one of the few “output-only” variables used by the standard utilities.

The <colon> is not allowed as an option character because that is not historical behavior, and it violates the Utility Syntax Guidelines. The <colon> is now specified to behave as in the KornShell version of the *getopts* utility; when used as the first character in the *optstring* operand, it disables diagnostics concerning missing option-arguments and unexpected option characters. This replaces the use of the *OPTERR* variable that was specified in an early proposal.

The formats of the diagnostic messages produced by the *getopts* utility and the *getopt()* function are not fully specified because implementations with superior (“friendlier”) formats objected to the formats used by some historical implementations. The standard developers considered it important that the information in the messages used be uniform between *getopts* and *getopt()*. Exact duplication of the messages might not be possible, particularly if a utility is built on another system that has a different *getopt()* function, but the messages must have specific information included so that the program name, invalid option character, and type of error can be distinguished by a user.

Only a rare application program intercepts a *getopts* standard error message and wants to parse it. Therefore, implementations are free to choose the most usable messages they can devise. The following formats are used by many historical implementations:

```
%s: illegal option -- %c\n", <program name>, <option character>

%s: option requires an argument -- %c\n", <program name>, \
    <option character>
```

getopts*Utilities*

90732 Historical shells with built-in versions of *getopt()* or *getopts* have used different formats,
90733 frequently not even indicating the option character found in error.

90734 **FUTURE DIRECTIONS**

90735 None.

90736 **SEE ALSO**

90737 [Section 2.5.2](#) (on page 2302)

90738 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

90739 XSH *getopt()*

90740 **CHANGE HISTORY**

90741 First released in Issue 4.

90742 **Issue 6**

90743 The normative text is reworded to avoid use of the term “must” for application requirements.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

grep — search a file for a pattern

SYNOPSIS

```
grep [-E|-F] [-c|-l|-q] [-insvx] -e pattern_list
    [-e pattern_list]... [-f pattern_file]... [file...]

grep [-E|-F] [-c|-l|-q] [-insvx] [-e pattern_list]...
    -f pattern_file [-f pattern_file]... [file...]

grep [-E|-F] [-c|-l|-q] [-insvx] pattern_list [file...]
```

DESCRIPTION

The *grep* utility shall search the input files, selecting lines matching one or more patterns; the types of patterns are controlled by the options specified. The patterns are specified by the *-e* option, *-f* option, or the *pattern_list* operand. The *pattern_list*'s value shall consist of one or more patterns separated by <newline> characters; the *pattern_file*'s contents shall consist of one or more patterns terminated by a <newline> character. By default, an input line shall be selected if any pattern, treated as an entire basic regular expression (BRE) as described in XBD Section 9.3 (on page 183), matches any part of the line excluding the terminating <newline>; a null BRE shall match every line. By default, each selected input line shall be written to the standard output.

Regular expression matching shall be based on text lines. Since a <newline> separates or terminates patterns (see the *-e* and *-f* options below), regular expressions cannot contain a <newline>. Similarly, since patterns are matched against individual lines (excluding the terminating <newline> characters) of the input, there is no way for a pattern to match a <newline> found in the input.

OPTIONS

The *grep* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

- E** Match using extended regular expressions. Treat each pattern specified as an ERE, as described in XBD Section 9.4 (on page 188). If any entire ERE pattern matches some part of an input line excluding the terminating <newline>, the line shall be matched. A null ERE shall match every line.
- F** Match using fixed strings. Treat each pattern specified as a string instead of a regular expression. If an input line contains any of the patterns as a contiguous sequence of bytes, the line shall be matched. A null string shall match every line.
- c** Write only a count of selected lines to standard output.
- e pattern_list** Specify one or more patterns to be used during the search for input. The application shall ensure that patterns in *pattern_list* are separated by a <newline>. A null pattern can be specified by two adjacent <newline> characters in *pattern_list*. Unless the *-E* or *-F* option is also specified, each pattern shall be treated as a BRE, as described in XBD Section 9.3 (on page 183). Multiple *-e* and *-f* options shall be accepted by the *grep* utility. All of the specified patterns shall be used when matching lines, but the order of evaluation is unspecified.
- f pattern_file** Read one or more patterns from the file named by the pathname *pattern_file*. Patterns in *pattern_file* shall be terminated by a <newline>. A null pattern can be specified by an empty line in *pattern_file*. Unless the *-E* or *-F* option is also

grep*Utilities*

90790		specified, each pattern shall be treated as a BRE, as described in XBD Section 9.3
90791		(on page 183).
90792	-i	Perform pattern matching in searches without regard to case; see XBD Section 9.2
90793		(on page 182).
90794	-l	(The letter ell.) Write only the names of files containing selected lines to standard
90795		output. Pathnames shall be written once per file searched. If the standard input is
90796		searched, a pathname of "(standard input)" shall be written, in the POSIX
90797		locale. In other locales, "standard input" may be replaced by something more
90798		appropriate in those locales.
90799	-n	Precede each output line by its relative line number in the file, each file starting at
90800		line 1. The line number counter shall be reset for each file processed.
90801	-q	Quiet. Nothing shall be written to the standard output, regardless of matching
90802		lines. Exit with zero status if an input line is selected.
90803	-s	Suppress the error messages ordinarily written for nonexistent or unreadable files.
90804		Other error messages shall not be suppressed.
90805	-v	Select lines not matching any of the specified patterns. If the -v option is not
90806		specified, selected lines shall be those that match any of the specified patterns.
90807	-x	Consider only input lines that use all characters in the line excluding the
90808		terminating <newline> to match an entire fixed string or regular expression to be
90809		matching lines.

OPERANDS

90810		The following operands shall be supported:
90811		
90812	<i>pattern_list</i>	Specify one or more patterns to be used during the search for input. This operand
90813		shall be treated as if it were specified as -e <i>pattern_list</i> .
90814	<i>file</i>	A pathname of a file to be searched for the patterns. If no <i>file</i> operands are
90815		specified, the standard input shall be used.

STDIN

90817		The standard input shall be used if no <i>file</i> operands are specified, and shall be used if a <i>file</i>
90818		operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
90819		the standard input shall not be used. See the INPUT FILES section.

INPUT FILES

90820		The input files shall be text files.

ENVIRONMENT VARIABLES

90822		The following environment variables shall affect the execution of <i>grep</i> :
90823		
90824	LANG	Provide a default value for the internationalization variables that are unset or null.
90825		(See XBD Section 8.2 (on page 174) for the precedence of internationalization
90826		variables used to determine the values of locale categories.)
90827	LC_ALL	If set to a non-empty string value, override the values of all the other
90828		internationalization variables.
90829	LC_COLLATE	
90830		Determine the locale for the behavior of ranges, equivalence classes, and multi-
90831		character collating elements within regular expressions.

- 90832 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 90833 characters (for example, single-byte as opposed to multi-byte characters in
 90834 arguments and input files) and the behavior of character classes within regular
 90835 expressions.
- 90836 **LC_MESSAGES**
 90837 Determine the locale that should be used to affect the format and contents of
 90838 diagnostic messages written to standard error.
- 90839 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 90840 **ASYNCHRONOUS EVENTS**
- 90841 Default.
- 90842 **STDOUT**
 90843 If the **-l** option is in effect, the following shall be written for each file containing at least one
 90844 selected input line:
 90845 "%s\n", <file>
- 90846 Otherwise, if more than one *file* argument appears, and **-q** is not specified, the *grep* utility shall
 90847 prefix each output line by:
 90848 "%s:", <file>
- 90849 The remainder of each output line shall depend on the other options specified:
- 90850 • If the **-c** option is in effect, the remainder of each output line shall contain:
 90851 "%d\n", <count>
 - 90852 • Otherwise, if **-c** is not in effect and the **-n** option is in effect, the following shall be written
 90853 to standard output:
 90854 "%d:", <line number>
 - 90855 • Finally, the following shall be written to standard output:
 90856 "%s", <selected line contents>
- 90857 **STDERR**
 90858 The standard error shall be used only for diagnostic messages.
- 90859 **OUTPUT FILES**
 90860 None.
- 90861 **EXTENDED DESCRIPTION**
 90862 None.
- 90863 **EXIT STATUS**
 90864 The following exit values shall be returned:
- 90865 0 One or more lines were selected.
 - 90866 1 No lines were selected.
 - 90867 >1 An error occurred.
- 90868 **CONSEQUENCES OF ERRORS**
 90869 If the **-q** option is specified, the exit status shall be zero if an input line is selected, even if an
 90870 error was detected. Otherwise, default actions shall be performed.

APPLICATION USAGE

Care should be taken when using characters in *pattern_list* that may also be meaningful to the command interpreter. It is safest to enclose the entire *pattern_list* argument in single-quotes:

```
'...'
```

The **-e** *pattern_list* option has the same effect as the *pattern_list* operand, but is useful when *pattern_list* begins with the <hyphen> delimiter. It is also useful when it is more convenient to provide multiple patterns as separate arguments.

Multiple **-e** and **-f** options are accepted and *grep* uses all of the patterns it is given while matching input text lines. (Note that the order of evaluation is not specified. If an implementation finds a null string as a pattern, it is allowed to use that pattern first, matching every line, and effectively ignore any other patterns.)

The **-q** option provides a means of easily determining whether or not a pattern (or string) exists in a group of files. When searching several files, it provides a performance improvement (because it can quit as soon as it finds the first match) and requires less care by the user in choosing the set of files to supply as arguments (because it exits zero if it finds a match even if *grep* detected an access or read error on earlier *file* operands).

EXAMPLES

1. To find all uses of the word "Posix" (in any case) in file **text.mm** and write with line numbers:

```
grep -i -n posix text.mm
```

2. To find all empty lines in the standard input:

```
grep ^$
```

or:

```
grep -v .
```

3. Both of the following commands print all lines containing strings "abc" or "def" or both:

```
grep -E 'abc|def'
```

```
grep -F 'abc
def'
```

4. Both of the following commands print all lines matching exactly "abc" or "def":

```
grep -E '^abc$|^def$'
```

```
grep -F -x 'abc
def'
```

RATIONALE

This *grep* has been enhanced in an upwards-compatible way to provide the exact functionality of the historical *egrep* and *fgrep* commands as well. It was the clear intention of the standard developers to consolidate the three *greps* into a single command.

The old *egrep* and *fgrep* commands are likely to be supported for many years to come as implementation extensions, allowing historical applications to operate unmodified.

Historical implementations usually silently ignored all but one of multiply-specified **-e** and **-f** options, but were not consistent as to which specification was actually used.

- 90912 The **-b** option was omitted from the OPTIONS section because block numbers are
90913 implementation-defined.
- 90914 The System V restriction on using **-** to mean standard input was omitted.
- 90915 A definition of action taken when given a null BRE or ERE is specified. This is an error
90916 condition in some historical implementations.
- 90917 The **-I** option previously indicated that its use was undefined when no files were explicitly
90918 named. This behavior was historical and placed an unnecessary restriction on future
90919 implementations. It has been removed.
- 90920 The historical BSD *grep* **-s** option practice is easily duplicated by redirecting standard output to
90921 **/dev/null**. The **-s** option required here is from System V.
- 90922 The **-x** option, historically available only with *fgrep*, is available here for all of the non-
90923 obsolescent versions.
- 90924 **FUTURE DIRECTIONS**
- 90925 None.
- 90926 **SEE ALSO**
- 90927 *sed*
- 90928 XBD [Chapter 8](#) (on page 173), [Chapter 9](#) (on page 181), [Section 12.2](#) (on page 215)
- 90929 **CHANGE HISTORY**
- 90930 First released in Issue 2.
- 90931 **Issue 6**
- 90932 The Open Group Corrigendum U029/5 is applied, correcting the SYNOPSIS.
- 90933 The normative text is reworded to avoid use of the term “must” for application requirements.
- 90934 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/28 is applied, correcting the examples
90935 using the *grep* **-F** option which did not match the normative description of the **-F** option.
- 90936 **Issue 7**
- 90937 Austin Group Interpretation 1003.1-2001 #092 is applied.
- 90938 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 90939 SD5-XCU-ERN-98 is applied, updating the STDOUT section.

hash

Utilities

90940 **NAME**

90941 hash — remember or report utility locations

90942 **SYNOPSIS**90943 hash [*utility*...]

90944 hash -r

90945 **DESCRIPTION**

90946 The *hash* utility shall affect the way the current shell environment remembers the locations of
 90947 utilities found as described in [Section 2.9.1.1](#) (on page 2317). Depending on the arguments
 90948 specified, it shall add utility locations to its list of remembered locations or it shall purge the
 90949 contents of the list. When no arguments are specified, it shall report on the contents of the list.

90950 Utilities provided as built-ins to the shell shall not be reported by *hash*.

90951 **OPTIONS**

90952 The *hash* utility shall conform to XBD [Section 12.2](#) (on page 215).

90953 The following option shall be supported:

90954 **-r** Forget all previously remembered utility locations.

90955 **OPERANDS**

90956 The following operand shall be supported:

90957 *utility* The name of a utility to be searched for and added to the list of remembered
 90958 locations. If *utility* contains one or more <slash> characters, the results are
 90959 unspecified.

90960 **STDIN**

90961 Not used.

90962 **INPUT FILES**

90963 None.

90964 **ENVIRONMENT VARIABLES**

90965 The following environment variables shall affect the execution of *hash*:

90966 **LANG** Provide a default value for the internationalization variables that are unset or null.
 90967 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 90968 variables used to determine the values of locale categories.)

90969 **LC_ALL** If set to a non-empty string value, override the values of all the other
 90970 internationalization variables.

90971 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 90972 characters (for example, single-byte as opposed to multi-byte characters in
 90973 arguments).

90974 **LC_MESSAGES**

90975 Determine the locale that should be used to affect the format and contents of
 90976 diagnostic messages written to standard error.

90977 **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

90978 **PATH** Determine the location of *utility*, as described in XBD [Chapter 8](#) (on page 173).

90979 **ASYNCHRONOUS EVENTS**

90980 Default.

90981 **STDOUT**

90982 The standard output of *hash* shall be used when no arguments are specified. Its format is
 90983 unspecified, but includes the pathname of each utility in the list of remembered locations for the
 90984 current shell environment. This list shall consist of those utilities named in previous *hash*
 90985 invocations that have been invoked, and may contain those invoked and found through the
 90986 normal command search process.

90987 **STDERR**

90988 The standard error shall be used only for diagnostic messages.

90989 **OUTPUT FILES**

90990 None.

90991 **EXTENDED DESCRIPTION**

90992 None.

90993 **EXIT STATUS**

90994 The following exit values shall be returned:

90995 0 Successful completion.

90996 >0 An error occurred.

90997 **CONSEQUENCES OF ERRORS**

90998 Default.

90999 **APPLICATION USAGE**

91000 Since *hash* affects the current shell execution environment, it is always provided as a shell
 91001 regular built-in. If it is called in a separate utility execution environment, such as one of the
 91002 following:

```
91003 nohup hash -r
91004 find . -type f | xargs hash
```

91005 it does not affect the command search process of the caller's environment.

91006 The *hash* utility may be implemented as an alias—for example, *alias -t -*, in which case utilities
 91007 found through normal command search are not listed by the *hash* command.

91008 The effects of *hash -r* can also be achieved portably by resetting the value of *PATH*; in the
 91009 simplest form, this can be:

91010 `PATH="$PATH"`

91011 The use of *hash* with *utility* names is unnecessary for most applications, but may provide a
 91012 performance improvement on a few implementations; normally, the hashing process is included
 91013 by default.

91014 **EXAMPLES**

91015 None.

91016 **RATIONALE**

91017 None.

hash*Utilities*91018 **FUTURE DIRECTIONS**

91019 None.

91020 **SEE ALSO**91021 [Section 2.9.1.1](#) (on page 2317)91022 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)91023 **CHANGE HISTORY**

91024 First released in Issue 2.

91025 **Issue 7**91026 The *hash* utility is moved from the XSI option to the Base.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

head91027 **NAME**

91028 head — copy the first part of files

91029 **SYNOPSIS**91030 head [-n *number*] [*file...*]91031 **DESCRIPTION**91032 The *head* utility shall copy its input files to the standard output, ending the output for each file at
91033 a designated point.91034 Copying shall end at the point in each input file indicated by the **-n *number*** option. The option-
91035 argument *number* shall be counted in units of lines.91036 **OPTIONS**91037 The *head* utility shall conform to XBD [Section 12.2](#) (on page 215).

91038 The following option shall be supported:

91039 **-n *number*** The first *number* lines of each input file shall be copied to standard output. The
91040 application shall ensure that the *number* option-argument is a positive decimal
91041 integer.91042 When a file contains less than *number* lines, it shall be copied to standard output in its entirety.
91043 This shall not be an error.91044 If no options are specified, *head* shall act as if **-n 10** had been specified.91045 **OPERANDS**

91046 The following operand shall be supported:

91047 *file* A pathname of an input file. If no *file* operands are specified, the standard input
91048 shall be used.91049 **STDIN**91050 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*
91051 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
91052 the standard input shall not be used. See the INPUT FILES section.91053 **INPUT FILES**

91054 Input files shall be text files, but the line length is not restricted to {LINE_MAX} bytes.

91055 **ENVIRONMENT VARIABLES**91056 The following environment variables shall affect the execution of *head*:91057 **LANG** Provide a default value for the internationalization variables that are unset or null.
91058 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
91059 variables used to determine the values of locale categories.)91060 **LC_ALL** If set to a non-empty string value, override the values of all the other
91061 internationalization variables.91062 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
91063 characters (for example, single-byte as opposed to multi-byte characters in
91064 arguments and input files).91065 **LC_MESSAGES**91066 Determine the locale that should be used to affect the format and contents of
91067 diagnostic messages written to standard error.

head

Utilities

91068 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

91069 **ASYNCHRONOUS EVENTS**

91070 Default.

91071 **STDOUT**

91072 The standard output shall contain designated portions of the input files.

91073 If multiple *file* operands are specified, *head* shall precede the output for each with the header:

91074 "`\n==> %s <==\n`", *<pathname>*

91075 except that the first header written shall not include the initial *<newline>*.

91076 **STDERR**

91077 The standard error shall be used only for diagnostic messages.

91078 **OUTPUT FILES**

91079 None.

91080 **EXTENDED DESCRIPTION**

91081 None.

91082 **EXIT STATUS**

91083 The following exit values shall be returned:

91084 0 Successful completion.

91085 >0 An error occurred.

91086 **CONSEQUENCES OF ERRORS**

91087 Default.

91088 **APPLICATION USAGE**

91089 None.

91090 **EXAMPLES**

91091 To write the first ten lines of all files (except those with a leading period) in the directory:

91092 `head -- *`

91093 **RATIONALE**

91094 Although it is possible to simulate *head* with *sed* 10q for a single file, the standard developers decided that the popularity of *head* on historical BSD systems warranted its inclusion alongside *tail*.

91097 POSIX.1-2008 version of *head* follows the Utility Syntax Guidelines. The `-n` option was added to this new interface so that *head* and *tail* would be more logically related. Earlier versions of this standard allowed a `-number` option. This form is no longer specified by POSIX.1-2008 but may be present in some implementations.

91101 There is no `-c` option (as there is in *tail*) because it is not historical practice and because other utilities in this volume of POSIX.1-2008 provide similar functionality.

91103 **FUTURE DIRECTIONS**

91104 None.

91105 **SEE ALSO**

91106 *sed*, *tail*

91107 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

91108 **CHANGE HISTORY**

91109 First released in Issue 4.

91110 **Issue 6**91111 The obsolescent –**number** form is removed.

91112 The normative text is reworded to avoid use of the term “must” for application requirements.

91113 The DESCRIPTION is updated to clarify that when a file contains less than the number of lines
91114 requested, the entire file is copied to standard output.91115 **Issue 7**

91116 Austin Group Interpretations 1003.1-2001 #027 and #092 are applied.

91117 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91118 The APPLICATION USAGE section is removed and the EXAMPLES section is corrected.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

iconv*Utilities*91119 **NAME**91120 `iconv` — codeset conversion91121 **SYNOPSIS**91122 `iconv [-cs] -f frommap -t tomap [file...]`91123 `iconv -f fromcode [-cs] [-t tocode] [file...]`91124 `iconv -t tocode [-cs] [-f fromcode] [file...]`91125 `iconv -l`91126 **DESCRIPTION**91127 The *iconv* utility shall convert the encoding of characters in *file* from one codeset to another and
91128 write the results to standard output.91129 When the options indicate that charmap files are used to specify the codesets (see **OPTIONS**),
91130 the codeset conversion shall be accomplished by performing a logical join on the symbolic
91131 character names in the two charmaps. The implementation need not support the use of charmap
91132 files for codeset conversion unless the `POSIX2_LOCALEDEF` symbol is defined on the system.91133 **OPTIONS**91134 The *iconv* utility shall conform to XBD [Section 12.2](#) (on page 215).

91135 The following options shall be supported:

91136 **-c** Omit any characters that are invalid in the codeset of the input file from the
91137 output. When **-c** is not used, the results of encountering invalid characters in the
91138 input stream (either those that are not characters in the codeset of the input file or
91139 that have no corresponding character in the codeset of the output file) shall be
91140 specified in the system documentation. The presence or absence of **-c** shall not
91141 affect the exit status of *iconv*.91142 **-f *fromcodeset***91143 Identify the codeset of the input file. The implementation shall recognize the
91144 following two forms of the *fromcodeset* option-argument:91145 *fromcode* The *fromcode* option-argument must not contain a <slash> character.
91146 It shall be interpreted as the name of one of the codeset descriptions
91147 provided by the implementation in an unspecified format. Valid
91148 values of *fromcode* are implementation-defined.91149 *frommap*91150 The *frommap* option-argument must contain a <slash> character. It
91151 shall be interpreted as the pathname of a charmap file as defined in
91152 XBD [Section 6.4](#) (on page 129). If the pathname does not represent a
valid, readable charmap file, the results are undefined.

91153 If this option is omitted, the codeset of the current locale shall be used.

91154 **-l** Write all supported *fromcode* and *tocode* values to standard output in an unspecified
91155 format.91156 **-s** Suppress any messages written to standard error concerning invalid characters.
91157 When **-s** is not used, the results of encountering invalid characters in the input
91158 stream (either those that are not valid characters in the codeset of the input file or
91159 that have no corresponding character in the codeset of the output file) shall be
91160 specified in the system documentation. The presence or absence of **-s** shall not
91161 affect the exit status of *iconv*.

91162 **-t** *tocodeset* Identify the codeset to be used for the output file. The implementation shall
91163 recognize the following two forms of the *tocodeset* option-argument:

91164 <i>tocode</i>	The semantics shall be equivalent to the -f <i>fromcode</i> option.
91165 <i>tomap</i>	The semantics shall be equivalent to the -f <i>frommap</i> option.

91166 If this option is omitted, the codeset of the current locale shall be used.

91167 If either **-f** or **-t** represents a charmap file, but the other does not (or is omitted), or both **-f** and
91168 **-t** are omitted, the results are undefined.

91169 OPERANDS

91170 The following operand shall be supported:

91171 *file* A pathname of an input file. If no *file* operands are specified, or if a *file* operand is
91172 *'-'*, the standard input shall be used.

91173 **STDIN**

91174 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
91175

91175 INPUT FILES

91176 The input file shall be a text file.

91177 ENVIRONMENT VARIABLES

91178 The following environment variables shall affect the execution of *iconv*:

91179 *LANG* Provide a default value for the internationalization variables that are unset or null.
91180 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
91181 variables used to determine the values of locale categories.)

91182	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other
91183		internationalization variables.

91184 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
91185 characters (for example, single-byte as opposed to multi-byte characters in
91186 arguments). During translation of the file, this variable is superseded by the use of
91187 the *fromcode* option-argument.

91188 *LC_MESSAGES*
91189 Determine the locale that should be used to affect the format and contents of
91190 diagnostic messages written to standard error.

```

91191  XSI      NLSPATH Determine the location of message catalogs for the processing of LC_MESSAGES.

```

91192 **ASYNCHRONOUS EVENTS**

91193 Default.

91194 **STDOUT**

When the `-I` option is used, the standard output shall contain all supported *fromcode* and *to code* values, written in an unspecified format.

91197 When the `-l` option is not used, the standard output shall contain the sequence of characters
91198 read from the input files, translated to the specified codeset. Nothing else shall be written to the
91199 standard output.

91200 STDERR

91201 The standard error shall be used only for diagnostic messages.

iconv*Utilities*91202 **OUTPUT FILES**

91203 None.

91204 **EXTENDED DESCRIPTION**

91205 None.

91206 **EXIT STATUS**

91207 The following exit values shall be returned:

91208 0 Successful completion.

91209 >0 An error occurred.

91210 **CONSEQUENCES OF ERRORS**

91211 Default.

91212 **APPLICATION USAGE**91213 The user must ensure that both charmap files use the same symbolic names for characters the
91214 two codesets have in common.91215 **EXAMPLES**91216 The following example converts the contents of file **mail.x400** from the ISO/IEC 6937:2001
91217 standard codeset to the ISO/IEC 8859-1:1998 standard codeset, and stores the results in file
91218 **mail.local**:91219 `iconv -f IS6937 -t IS8859 mail.x400 > mail.local`91220 **RATIONALE**91221 The *iconv* utility can be used portably only when the user provides two charmap files as option-
91222 arguments. This is because a single charmap provided by the user cannot reliably be joined with
91223 the names in a system-provided character set description. The valid values for *fromcode* and
91224 *tocode* are implementation-defined and do not have to have any relation to the charmap
91225 mechanisms. As an aid to interactive users, the `-l` option was adopted from the Plan 9 operating
91226 system. It writes information concerning these implementation-defined values. The format is
91227 unspecified because there are many possible useful formats that could be chosen, such as a
91228 matrix of valid combinations of *fromcode* and *tocode*. The `-l` option is not intended for shell script
91229 usage; conforming applications will have to use charmaps.91230 **FUTURE DIRECTIONS**

91231 None.

91232 **SEE ALSO**91233 *gencat*

91234 XBD Section 6.4 (on page 129), Chapter 8 (on page 173), Section 12.2 (on page 215)

91235 **CHANGE HISTORY**

91236 First released in Issue 3.

91237 **Issue 6**91238 This utility has been rewritten to align with the IEEE P1003.2b draft standard. Specifically, the
91239 ability to use charmap files for conversion has been added.91240 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/29 is applied, making changes to address
91241 inconsistencies with the *iconv()* function in the System Interfaces volume of POSIX.1-2008.

91242 **Issue 7**91243 Austin Group Interpretation 1003.1-2001 #206 is applied, correcting the *tomap* option.

91244 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

id

Utilities

91245 **NAME**91246 **id** — return user identity91247 **SYNOPSIS**91248 **id** [*user*]91249 **id** -G [-n] [*user*]91250 **id** -g [-nr] [*user*]91251 **id** -u [-nr] [*user*]91252 **DESCRIPTION**

91253 If no *user* operand is provided, the *id* utility shall write the user and group IDs and the
 91254 corresponding user and group names of the invoking process to standard output. If the effective
 91255 and real IDs do not match, both shall be written. If multiple groups are supported by the
 91256 underlying system (see the description of {NGROUPS_MAX} in the System Interfaces volume of
 91257 POSIX.1-2008), the supplementary group affiliations of the invoking process shall also be
 91258 written.

91259 If a *user* operand is provided and the process has appropriate privileges, the user and group IDs
 91260 of the selected user shall be written. In this case, effective IDs shall be assumed to be identical to
 91261 real IDs. If the selected user has more than one allowable group membership listed in the group
 91262 database, these shall be written in the same manner as the supplementary groups described in
 91263 the preceding paragraph.

91264 **OPTIONS**91265 The *id* utility shall conform to XBD Section 12.2 (on page 215).

91266 The following options shall be supported:

91267 **-G** Output all different group IDs (effective, real, and supplementary) only, using the
 91268 format "%u\n". If there is more than one distinct group affiliation, output each
 91269 such affiliation, using the format " %u", before the <newline> is output.

91270 **-g** Output only the effective group ID, using the format "%u\n".

91271 **-n** Output the name in the format "%s" instead of the numeric ID using the format
 91272 "%u".

91273 **-r** Output the real ID instead of the effective ID.

91274 **-u** Output only the effective user ID, using the format "%u\n".

91275 **OPERANDS**

91276 The following operand shall be supported:

91277 *user* The login name for which information is to be written.

91278 **STDIN**

91279 Not used.

91280 **INPUT FILES**

91281 None.

91282 **ENVIRONMENT VARIABLES**91283 The following environment variables shall affect the execution of *id*:

91284 **LANG** Provide a default value for the internationalization variables that are unset or null.
 91285 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 91286 variables used to determine the values of locale categories.)

91287	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
91288		
91289	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
91290		
91291		
91292	<i>LC_MESSAGES</i>	
91293		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.
91294		
91295		
91296	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
91297	ASYNCHRONOUS EVENTS	
91298		Default.
91299	STDOUT	
91300		The following formats shall be used when the <i>LC_MESSAGES</i> locale category specifies the POSIX locale. In other locales, the strings <i>uid</i> , <i>gid</i> , <i>euid</i> , <i>egid</i> , and <i>groups</i> may be replaced with more appropriate strings corresponding to the locale.
91301		
91302		
91303		"uid=%u(%s) gid=%u(%s)\n", <real user ID>, <user-name>, 91304 <real group ID>, <group-name>
91305		If the effective and real user IDs do not match, the following shall be inserted immediately before the '\n' character in the previous format:
91306		
91307		" euid=%u(%s) "
91308		with the following arguments added at the end of the argument list:
91309		<effective user ID>, <effective user-name>
91310		If the effective and real group IDs do not match, the following shall be inserted directly before the '\n' character in the format string (and after any addition resulting from the effective and real user IDs not matching):
91311		
91312		
91313		" egid=%u(%s) "
91314		with the following arguments added at the end of the argument list:
91315		<effective group-ID>, <effective group name>
91316		If the process has supplementary group affiliations or the selected user is allowed to belong to multiple groups, the first shall be added directly before the <newline> in the format string:
91317		
91318		" groups=%u(%s) "
91319		with the following arguments added at the end of the argument list:
91320		<supplementary group ID>, <supplementary group name>
91321		and the necessary number of the following added after that for any remaining supplementary group IDs:
91322		
91323		", %u(%s) "
91324		and the necessary number of the following arguments added at the end of the argument list:
91325		<supplementary group ID>, <supplementary group name>
91326		If any of the user ID, group ID, effective user ID, effective group ID, or supplementary/multiple

91327 group IDs cannot be mapped by the system into printable user or group names, the
 91328 corresponding "(%s)" and *name* argument shall be omitted from the corresponding format
 91329 string.

91330 When any of the options are specified, the output format shall be as described in the OPTIONS
 91331 section.

91332 **STDERR**

91333 The standard error shall be used only for diagnostic messages.

91334 **OUTPUT FILES**

91335 None.

91336 **EXTENDED DESCRIPTION**

91337 None.

91338 **EXIT STATUS**

91339 The following exit values shall be returned:

91340 0 Successful completion.

91341 >0 An error occurred.

91342 **CONSEQUENCES OF ERRORS**

91343 Default.

91344 **APPLICATION USAGE**

91345 Output produced by the **-G** option and by the default case could potentially produce very long
 91346 lines on systems that support large numbers of supplementary groups. (On systems with user
 91347 and group IDs that are 32-bit integers and with group names with a maximum of 8 bytes per
 91348 name, 93 supplementary groups plus distinct effective and real group and user IDs could
 91349 theoretically overflow the 2048-byte {LINE_MAX} text file line limit on the default output case.
 91350 It would take about 186 supplementary groups to overflow the 2048-byte barrier using *id -G*).
 91351 This is not expected to be a problem in practice, but in cases where it is a concern, applications
 91352 should consider using *fold -s* before post-processing the output of *id*.

91353 **EXAMPLES**

91354 None.

91355 **RATIONALE**

91356 The functionality provided by the 4 BSD *groups* utility can be simulated using:

91357 `id -Gn [ user ]`

91358 The 4 BSD command *groups* was considered, but it was not included because it did not provide
 91359 the functionality of the *id* utility of the SVID. Also, it was thought that it would be easier to
 91360 modify *id* to provide the additional functionality necessary to systems with multiple groups than
 91361 to invent another command.

91362 The options **-u**, **-g**, **-n**, and **-r** were added to ease the use of *id* with shell commands
 91363 substitution. Without these options it is necessary to use some preprocessor such as *sed* to select
 91364 the desired piece of information. Since output such as that produced by:

91365 `id -u -n`

91366 is frequently wanted, it seemed desirable to add the options.

91367 **FUTURE DIRECTIONS**

91368 None.

91369 **SEE ALSO**91370 *fold, logname, who*91371 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)91372 XSH [getgid\(\)](#), [getgroups\(\)](#), [getuid\(\)](#)91373 **CHANGE HISTORY**

91374 First released in Issue 2.

91375 **Issue 7**

91376 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

ipcrm

Utilities

91377 **NAME**91378 **ipcrm** — remove an XSI message queue, semaphore set, or shared memory segment identifier91379 **SYNOPSIS**91380 XSI `ipcrm [-q msgid|-Q msgkey|-s semid|-S semkey|-m shmid|-M shmkey] ...`91381 **DESCRIPTION**91382 The *ipcrm* utility shall remove zero or more message queues, semaphore sets, or shared memory
91383 segments. The interprocess communication facilities to be removed are specified by the options.91384 Only a user with appropriate privileges shall be allowed to remove an interprocess
91385 communication facility that was not created by or owned by the user invoking *ipcrm*.91386 **OPTIONS**91387 The *ipcrm* utility shall conform to XBD [Section 12.2](#) (on page 215).

91388 The following options shall be supported:

- 91389 **-q msgid** Remove the message queue identifier *msgid* from the system and destroy the
91390 message queue and data structure associated with it.
- 91391 **-m shmid** Remove the shared memory identifier *shmid* from the system. The shared memory
91392 segment and data structure associated with it shall be destroyed after the last
91393 detach.
- 91394 **-s semid** Remove the semaphore identifier *semid* from the system and destroy the set of
91395 semaphores and data structure associated with it.
- 91396 **-Q msgkey** Remove the message queue identifier, created with key *msgkey*, from the system
91397 and destroy the message queue and data structure associated with it.
- 91398 **-M shmkey** Remove the shared memory identifier, created with key *shmkey*, from the system.
91399 The shared memory segment and data structure associated with it shall be
91400 destroyed after the last detach.
- 91401 **-S semkey** Remove the semaphore identifier, created with key *semkey*, from the system and
91402 destroy the set of semaphores and data structure associated with it.

91403 **OPERANDS**

91404 None.

91405 **STDIN**

91406 Not used.

91407 **INPUT FILES**

91408 None.

91409 **ENVIRONMENT VARIABLES**91410 The following environment variables shall affect the execution of *ipcrm*:

- 91411 **LANG** Provide a default value for the internationalization variables that are unset or null.
91412 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
91413 variables used to determine the values of locale categories.)
- 91414 **LC_ALL** If set to a non-empty string value, override the values of all the other
91415 internationalization variables.
- 91416 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
91417 characters (for example, single-byte as opposed to multi-byte characters in
91418 arguments).

91419	<i>LC_MESSAGES</i>	
91420		Determine the locale that should be used to affect the format and contents of
91421		diagnostic messages written to standard error.
91422	<i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
91423	ASYNCHRONOUS EVENTS	
91424		Default.
91425	STDOUT	
91426		Not used.
91427	STDERR	
91428		The standard error shall be used only for diagnostic messages.
91429	OUTPUT FILES	
91430		None.
91431	EXTENDED DESCRIPTION	
91432		None.
91433	EXIT STATUS	
91434		The following exit values shall be returned:
91435	0	Successful completion.
91436	>0	An error occurred.
91437	CONSEQUENCES OF ERRORS	
91438		Default.
91439	APPLICATION USAGE	
91440		None.
91441	EXAMPLES	
91442		None.
91443	RATIONALE	
91444		None.
91445	FUTURE DIRECTIONS	
91446		None.
91447	SEE ALSO	
91448		<i>ipcs</i>
91449		XBD Chapter 8 (on page 173), Section 12.2 (on page 215)
91450		XSH msgctl() , semctl() , shmctl()
91451	CHANGE HISTORY	
91452		First released in Issue 5.
91453	Issue 7	
91454		SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

ipcs

Utilities

91455 **NAME**91456 **ipcs** — report XSI interprocess communication facilities status91457 **SYNOPSIS**91458 XSI `ipcs [-qms] [-a|-bcopt]`91459 **DESCRIPTION**91460 The *ipcs* utility shall write information about active interprocess communication facilities.

91461 Without options, information shall be written in short format for message queues, shared
 91462 memory segments, and semaphore sets that are currently active in the system. Otherwise, the
 91463 information that is displayed is controlled by the options specified.

91464 **OPTIONS**91465 The *ipcs* utility shall conform to XBD [Section 12.2](#) (on page 215).91466 The *ipcs* utility accepts the following options:91467 **-q** Write information about active message queues.91468 **-m** Write information about active shared memory segments.91469 **-s** Write information about active semaphore sets.

91470 If **-q**, **-m**, or **-s** are specified, only information about those facilities shall be written. If none of
 91471 these three are specified, information about all three shall be written subject to the following
 91472 options:

91473 **-a** Use all print options. (This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.)

91474 **-b** Write information on maximum allowable size. (Maximum number of bytes in
 91475 messages on queue for message queues, size of segments for shared memory, and
 91476 number of semaphores in each set for semaphores.)

91477 **-c** Write creator's user name and group name; see below.

91478 **-o** Write information on outstanding usage. (Number of messages on queue and total
 91479 number of bytes in messages on queue for message queues, and number of
 91480 processes attached to shared memory segments.)

91481 **-p** Write process number information. (Process ID of the last process to send a
 91482 message and process ID of the last process to receive a message on message
 91483 queues, process ID of the creating process, and process ID of the last process to
 91484 attach or detach on shared memory segments.)

91485 **-t** Write time information. (Time of the last control operation that changed the access
 91486 permissions for all facilities, time of the last *msgsnd()* and *msgrcv()* operations on
 91487 message queues, time of the last *shmat()* and *shmdt()* operations on shared
 91488 memory, and time of the last *semop()* operation on semaphores.)

91489 **OPERANDS**

91490 None.

91491 **STDIN**

91492 Not used.

91493 **INPUT FILES**

- 91494 • The group database

- The user database

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ipcs*:

LANG	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
LC_ALL	If set to a non-empty string value, override the values of all the other internationalization variables.
LC_CTYPE	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
LC_MESSAGES	Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
NLSPATH	Determine the location of message catalogs for the processing of LC_MESSAGES.
TZ	Determine the timezone for the date and time strings written by <i>ipcs</i> . If TZ is unset or null, an unspecified default timezone shall be used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

An introductory line shall be written with the format:

```
"IPC status from %s as of %s\n", <source>, <date>
```

where <source> indicates the source used to gather the statistics and <date> is the information that would be produced by the *date* command when invoked in the POSIX locale.

The *ipcs* utility then shall create up to three reports depending upon the *-q*, *-m*, and *-s* options. The first report shall indicate the status of message queues, the second report shall indicate the status of shared memory segments, and the third report shall indicate the status of semaphore sets.

If the corresponding facility is not installed or has not been used since the last reboot, then the report shall be written out in the format:

```
"%s facility not in system.\n", <facility>
```

where <facility> is *Message Queue*, *Shared Memory*, or *Semaphore*, as appropriate. If the facility has been installed and has been used since the last reboot, column headings separated by one or more <space> characters and followed by a <newline> shall be written as indicated below followed by the facility name written out using the format:

```
"%s:\n", <facility>
```

where <facility> is *Message Queues*, *Shared Memory*, or *Semaphores*, as appropriate. On the second and third reports the column headings need not be written if the last column headings written already provide column headings for all information in that report.

The column headings provided in the first column below and the meaning of the information in those columns shall be given in order below; the letters in parentheses indicate the options that shall cause the corresponding column to appear; "all" means that the column shall always appear. Each column is separated by one or more <space> characters. Note that these options

91538		only determine what information is provided for each report; they do not determine which
91539		reports are written.
91540	T (all)	Type of facility:
91541		q Message queue.
91542		m Shared memory segment.
91543		s Semaphore.
91544		This field is a single character written using the format %c.
91545	ID (all)	The identifier for the facility entry. This field shall be written using the format
91546		%d.
91547	KEY (all)	The key used as an argument to <i>msgget()</i> , <i>semget()</i> , or <i>shmget()</i> to create the
91548		facility entry.
91549		Note: The key of a shared memory segment is changed to IPC_PRIVATE when the
91550		segment has been removed until all processes attached to the segment
91551		detach it.
91552		This field shall be written using the format 0x%x.
91553	MODE (all)	The facility access modes and flags. The mode shall consist of 11 characters
91554		that are interpreted as follows.
91555		The first character shall be:
91556		S If a process is waiting on a <i>msgsnd()</i> operation.
91557		– If the above is not true.
91558		The second character shall be:
91559		R If a process is waiting on a <i>msgrcv()</i> operation.
91560		C or – If the associated shared memory segment is to be cleared when the
91561		first attach operation is executed.
91562		– If none of the above is true.
91563		The next nine characters shall be interpreted as three sets of three bits each.
91564		The first set refers to the owner's permissions; the next to permissions of
91565		others in the usergroup of the facility entry; and the last to all others. Within
91566		each set, the first character indicates permission to read, the second character
91567		indicates permission to write or alter the facility entry, and the last character is
91568		a minus-sign ('-').
91569		The permissions shall be indicated as follows:
91570		r If read permission is granted.
91571		w If write permission is granted.
91572		a If alter permission is granted.
91573		– If the indicated permission is not granted.
91574		The first character following the permissions specifies if there is an alternate or
91575		additional access control method associated with the facility. If there is no
91576		alternate or additional access control method associated with the facility, a
91577		single <space> shall be written; otherwise, another printable character is

91578		written.
91579	OWNER (all)	The user name of the owner of the facility entry. If the user name of the owner
91580		is found in the user database, at least the first eight column positions of the
91581		name shall be written using the format %s. Otherwise, the user ID of the
91582		owner shall be written using the format %d.
91583	GROUP (all)	The group name of the owner of the facility entry. If the group name of the
91584		owner is found in the group database, at least the first eight column positions
91585		of the name shall be written using the format %s. Otherwise, the group ID of
91586		the owner shall be written using the format %d.
91587	The following nine columns shall be only written out for message queues:	
91588	CREATOR (a,c)	The user name of the creator of the facility entry. If the user name of the
91589		creator is found in the user database, at least the first eight column positions of
91590		the name shall be written using the format %s. Otherwise, the user ID of the
91591		creator shall be written using the format %d.
91592	CGROUP (a,c)	The group name of the creator of the facility entry. If the group name of the
91593		creator is found in the group database, at least the first eight column positions
91594		of the name shall be written using the format %s. Otherwise, the group ID of
91595		the creator shall be written using the format %d.
91596	CBYTES (a,o)	The number of bytes in messages currently outstanding on the associated
91597		message queue. This field shall be written using the format %d.
91598	QNUM (a,o)	The number of messages currently outstanding on the associated message
91599		queue. This field shall be written using the format %d.
91600	QBYTES (a,b)	The maximum number of bytes allowed in messages outstanding on the
91601		associated message queue. This field shall be written using the format %d.
91602	LSPID (a,p)	The process ID of the last process to send a message to the associated queue.
91603		This field shall be written using the format:
91604		"%d", <pid>
91605		where <pid> is 0 if no message has been sent to the corresponding message
91606		queue; otherwise, <pid> shall be the process ID of the last process to send a
91607		message to the queue.
91608	LRPID (a,p)	The process ID of the last process to receive a message from the associated
91609		queue. This field shall be written using the format:
91610		"%d", <pid>
91611		where <pid> is 0 if no message has been received from the corresponding
91612		message queue; otherwise, <pid> shall be the process ID of the last process to
91613		receive a message from the queue.
91614	STIME (a,t)	The time the last message was sent to the associated queue. If a message has
91615		been sent to the corresponding message queue, the hour, minute, and second
91616		of the last time a message was sent to the queue shall be written using the
91617		format %d:%2.2d:%2.2d. Otherwise, the format "no-entry" shall be
91618		written.
91619	RTIME (a,t)	The time the last message was received from the associated queue. If a
91620		message has been received from the corresponding message queue, the hour,
91621		minute, and second of the last time a message was received from the queue

91622		shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format
91623		" no-entry" shall be written.
91624	The following eight columns shall be only written out for shared memory segments.	
91625	CREATOR (a,c)	The user of the creator of the facility entry. If the user name of the creator is
91626		found in the user database, at least the first eight column positions of the
91627		name shall be written using the format %s. Otherwise, the user ID of the
91628		creator shall be written using the format %d.
91629	CGROUP (a,c)	The group name of the creator of the facility entry. If the group name of the
91630		creator is found in the group database, at least the first eight column positions
91631		of the name shall be written using the format %s. Otherwise, the group ID of
91632		the creator shall be written using the format %d.
91633	NATTCH (a,o)	The number of processes attached to the associated shared memory segment.
91634		This field shall be written using the format %d.
91635	SEGSZ (a,b)	The size of the associated shared memory segment. This field shall be written
91636		using the format %d.
91637	CPID (a,p)	The process ID of the creator of the shared memory entry. This field shall be
91638		written using the format %d.
91639	LPID (a,p)	The process ID of the last process to attach or detach the shared memory
91640		segment. This field shall be written using the format:
91641		"%d", <pid>
91642		where <pid> is 0 if no process has attached the corresponding shared memory
91643		segment; otherwise, <pid> shall be the process ID of the last process to attach
91644		or detach the segment.
91645	ATIME (a,t)	The time the last attach on the associated shared memory segment was
91646		completed. If the corresponding shared memory segment has ever been
91647		attached, the hour, minute, and second of the last time the segment was
91648		attached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the
91649		format " no-entry" shall be written.
91650	DTIME (a,t)	The time the last detach on the associated shared memory segment was
91651		completed. If the corresponding shared memory segment has ever been
91652		detached, the hour, minute, and second of the last time the segment was
91653		detached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the
91654		format " no-entry" shall be written.
91655	The following four columns shall be only written out for semaphore sets:	
91656	CREATOR (a,c)	The user of the creator of the facility entry. If the user name of the creator is
91657		found in the user database, at least the first eight column positions of the
91658		name shall be written using the format %s. Otherwise, the user ID of the
91659		creator shall be written using the format %d.
91660	CGROUP (a,c)	The group name of the creator of the facility entry. If the group name of the
91661		creator is found in the group database, at least the first eight column positions
91662		of the name shall be written using the format %s. Otherwise, the group ID of
91663		the creator shall be written using the format %d.

91664	NSEMS (a,b)	The number of semaphores in the set associated with the semaphore entry.
91665		This field shall be written using the format %d.
91666	OTIME (a,t)	The time the last semaphore operation on the set associated with the semaphore entry was completed. If a semaphore operation has ever been performed on the corresponding semaphore set, the hour, minute, and second of the last semaphore operation on the semaphore set shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be written.
91667		
91668		
91669		
91670		
91671		
91672		The following column shall be written for all three reports when it is requested:
91673	CTIME (a,t)	The time the associated entry was created or changed. The hour, minute, and second of the time when the associated entry was created shall be written using the format %d:%2.2d:%2.2d.
91674		
91675		
91676	STDERR	
91677		The standard error shall be used only for diagnostic messages.
91678	OUTPUT FILES	
91679		None.
91680	EXTENDED DESCRIPTION	
91681		None.
91682	EXIT STATUS	
91683		The following exit values shall be returned:
91684	0	Successful completion.
91685	>0	An error occurred.
91686	CONSEQUENCES OF ERRORS	
91687		Default.
91688	APPLICATION USAGE	
91689		Things can change while <i>ipcs</i> is running; the information it gives is guaranteed to be accurate only when it was retrieved.
91690		
91691	EXAMPLES	
91692		None.
91693	RATIONALE	
91694		None.
91695	FUTURE DIRECTIONS	
91696		None.
91697	SEE ALSO	
91698		<i>ipcrm</i>
91699		XBD Chapter 8 (on page 173), Section 12.2 (on page 215)
91700		XSH <i>msgrcv()</i> , <i>msgsnd()</i> , <i>semget()</i> , <i>semop()</i> , <i>shmat()</i> , <i>shmdt()</i> , <i>shmget()</i>
91701	CHANGE HISTORY	
91702		First released in Issue 5.

91703 **Issue 6**

91704 The Open Group Corrigendum U020/1 is applied, correcting the SYNOPSIS.

91705 The Open Group Corrigenda U032/1 and U032/2 are applied, clarifying the output format.

91706 The Open Group Base Resolution bwg98-004 is applied.

91707 **Issue 7**

91708 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91709 SD5-XCU-ERN-139 is applied, adding the *ipcrm* utility to the SEE ALSO section.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

jobs

91710 NAME

91711 jobs — display status of jobs in the current session

91712 SYNOPSIS

91713 UP jobs [-l|-p] [*job_id*...]

91714 DESCRIPTION

91715 The *jobs* utility shall display the status of jobs that were started in the current shell environment;
91716 see [Section 2.12](#) (on page 2331).

91717 When *jobs* reports the termination status of a job, the shell shall remove its process ID from the
91718 list of those “known in the current shell execution environment”; see [Section 2.9.3.1](#) (on page
91719 2319).

91720 OPTIONS

91721 The *jobs* utility shall conform to XBD [Section 12.2](#) (on page 215).

91722 The following options shall be supported:

91723 -l (The letter ell.) Provide more information about each job listed. This information
91724 shall include the job number, current job, process group ID, state, and the
91725 command that formed the job.

91726 -p Display only the process IDs for the process group leaders of the selected jobs.

91727 By default, the *jobs* utility shall display the status of all stopped jobs, running background jobs
91728 and all jobs whose status has changed and have not been reported by the shell.

91729 OPERANDS

91730 The following operand shall be supported:

91731 *job_id* Specifies the jobs for which the status is to be displayed. If no *job_id* is given, the
91732 status information for all jobs shall be displayed. The format of *job_id* is described
91733 in XBD [Section 3.203](#) (on page 65).

91734 STDIN

91735 Not used.

91736 INPUT FILES

91737 None.

91738 ENVIRONMENT VARIABLES

91739 The following environment variables shall affect the execution of *jobs*:

91740 LANG Provide a default value for the internationalization variables that are unset or null.
91741 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
91742 variables used to determine the values of locale categories.)

91743 LC_ALL If set to a non-empty string value, override the values of all the other
91744 internationalization variables.

91745 LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as
91746 characters (for example, single-byte as opposed to multi-byte characters in
91747 arguments).

91748 LC_MESSAGES

91749 Determine the locale that should be used to affect the format and contents of
91750 diagnostic messages written to standard error and informative messages written to
91751 standard output.

jobs

Utilities

91752 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

91753 ASYNCHRONOUS EVENTS

91754 Default.

91755 STDOUT

91756 If the **-p** option is specified, the output shall consist of one line for each process ID:

91757 "%d\n", <process ID>

91758 Otherwise, if the **-l** option is not specified, the output shall be a series of lines of the form:

91759 "[%d] %c %s %s\n", <job-number>, <current>, <state>, <command>

91760 where the fields shall be as follows:

91761 <current> The character '+' identifies the job that would be used as a default for the *fg* or *bg*
 91762 utilities; this job can also be specified using the *job_id* %+ or "%%". The character
 91763 '-' identifies the job that would become the default if the current default job were
 91764 to exit; this job can also be specified using the *job_id* %-. For other jobs, this field is
 91765 a <space>. At most one job can be identified with '+' and at most one job can be
 91766 identified with '-'. If there is any suspended job, then the current job shall be a
 91767 suspended job. If there are at least two suspended jobs, then the previous job also
 91768 shall be a suspended job.

91769 <job-number> A number that can be used to identify the process group to the *wait*, *fg*, *bg*, and *kill*
 91770 utilities. Using these utilities, the job can be identified by prefixing the job number
 91771 with '% '.

91772 <state> One of the following strings (in the POSIX locale):

91773 **Running** Indicates that the job has not been suspended by a signal and has not
 91774 exited.

91775 **Done** Indicates that the job completed and returned exit status zero.

91776 **Done(code)** Indicates that the job completed normally and that it exited with the
 91777 specified non-zero exit status, *code*, expressed as a decimal number.

91778 **Stopped** Indicates that the job was suspended by the SIGTSTP signal.

91779 **Stopped (SIGTSTP)**

91780 Indicates that the job was suspended by the SIGTSTP signal.

91781 **Stopped (SIGSTOP)**

91782 Indicates that the job was suspended by the SIGSTOP signal.

91783 **Stopped (SIGTTIN)**

91784 Indicates that the job was suspended by the SIGTTIN signal.

91785 **Stopped (SIGTTOU)**

91786 Indicates that the job was suspended by the SIGTTOU signal.

91787 The implementation may substitute the string **Suspended** in place of **Stopped**. If
 91788 the job was terminated by a signal, the format of <state> is unspecified, but it shall
 91789 be visibly distinct from all of the other <state> formats shown here and shall
 91790 indicate the name or description of the signal causing the termination.

91791 <command> The associated command that was given to the shell.

91792 If the **-l** option is specified, a field containing the process group ID shall be inserted before the
 91793 <state> field. Also, more processes in a process group may be output on separate lines, using

91794 only the process ID and *<command>* fields.

91795 **STDERR**

91796 The standard error shall be used only for diagnostic messages.

91797 **OUTPUT FILES**

91798 None.

91799 **EXTENDED DESCRIPTION**

91800 None.

91801 **EXIT STATUS**

91802 The following exit values shall be returned:

91803 0 Successful completion.

91804 >0 An error occurred.

91805 **CONSEQUENCES OF ERRORS**

91806 Default.

91807 **APPLICATION USAGE**

91808 The *-p* option is the only portable way to find out the process group of a job because different
91809 implementations have different strategies for defining the process group of the job. Usage such
91810 as *\$(jobs -p)* provides a way of referring to the process group of the job in an implementation-
91811 independent way.

91812 The *jobs* utility does not work as expected when it is operating in its own utility execution
91813 environment because that environment has no applicable jobs to manipulate. See the
91814 APPLICATION USAGE section for *bg*. For this reason, *jobs* is generally implemented as a shell
91815 regular built-in.

91816 **EXAMPLES**

91817 None.

91818 **RATIONALE**

91819 Both "%%" and "%+" are used to refer to the current job. Both forms are of equal validity—the
91820 "%%" mirroring "\$\$" and "%+" mirroring the output of *jobs*. Both forms reflect historical
91821 practice of the KornShell and the C shell with job control.

91822 The job control features provided by *bg*, *fg*, and *jobs* are based on the KornShell. The standard
91823 developers examined the characteristics of the C shell versions of these utilities and found that
91824 differences exist. Despite widespread use of the C shell, the KornShell versions were selected for
91825 this volume of POSIX.1-2008 to maintain a degree of uniformity with the rest of the KornShell
91826 features selected (such as the very popular command line editing features).

91827 The *jobs* utility is not dependent on the job control option, as are the seemingly related *bg* and *fg*
91828 utilities because *jobs* is useful for examining background jobs, regardless of the condition of job
91829 control. When the user has invoked a *set +m* command and job control has been turned off, *jobs*
91830 can still be used to examine the background jobs associated with that current session. Similarly,
91831 *kill* can then be used to kill background jobs with *kill %<background job number>*.

91832 The output for terminated jobs is left unspecified to accommodate various historical systems.
91833 The following formats have been witnessed:

91834 1. **Killed**(*signal name*)

91835 2. *signal name*

jobs*Utilities*

91836 3. *signal name*(**coredump**)

91837 4. *signal description*– **core dumped**

91838 Most users should be able to understand these formats, although it means that applications have
91839 trouble parsing them.

91840 The calculation of job IDs was not described since this would suggest an implementation, which
91841 may impose unnecessary restrictions.

91842 In an early proposal, a **-n** option was included to “Display the status of jobs that have changed,
91843 exited, or stopped since the last status report”. It was removed because the shell always writes
91844 any changed status of jobs before each prompt.

91845 **FUTURE DIRECTIONS**

91846 None.

91847 **SEE ALSO**

91848 [Section 2.12](#) (on page 2331), *bg*, *fg*, *kill*, *wait*

91849 XBD [Section 3.203](#) (on page 65), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

91850 **CHANGE HISTORY**

91851 First released in Issue 4.

91852 **Issue 6**

91853 This utility is marked as part of the User Portability Utilities option.

91854 The JC shading is removed as job control is mandatory in this version.

91855 **Issue 7**

91856 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

NAME

join — relational database operator

SYNOPSIS

```
join [-a file_number|-v file_number] [-e string] [-o list] [-t char]
    [-1 field] [-2 field] file1 file2
```

DESCRIPTION

The *join* utility shall perform an equality join on the files *file1* and *file2*. The joined files shall be written to the standard output.

The join field is a field in each file on which the files are compared. The *join* utility shall write one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line by default shall consist of the join field, then the remaining fields from *file1*, then the remaining fields from *file2*. This format can be changed by using the **-o** option (see below). The **-a** option can be used to add unmatched lines to the output. The **-v** option can be used to output only unmatched lines.

The files *file1* and *file2* shall be ordered in the collating sequence of *sort -b* on the fields on which they shall be joined, by default the first in each line. All selected output shall be written in the same collating sequence.

The default input field separators shall be <blank> characters. In this case, multiple separators shall count as one field separator, and leading separators shall be ignored. The default output field separator shall be a <space>.

The field separator and collating sequence can be changed by using the **-t** option (see below).

If the same key appears more than once in either file, all combinations of the set of remaining fields in *file1* and the set of remaining fields in *file2* are output in the order of the lines encountered.

If the input files are not in the appropriate collating sequence, the results are unspecified.

OPTIONS

The *join* utility shall conform to XBD [Section 12.2](#) (on page 215).

The following options shall be supported:

-a file_number

Produce a line for each unpairable line in file *file_number*, where *file_number* is 1 or 2, in addition to the default output. If both **-a1** and **-a2** are specified, all unpairable lines shall be output.

-e string

Replace empty output fields in the list selected by **-o** with the string *string*.

-o list

Construct the output line to comprise the fields specified in *list*, each element of which shall have one of the following two forms:

1. *file_number.field*, where *file_number* is a file number and *field* is a decimal integer field number
2. 0 (zero), representing the join field

The elements of *list* shall be either <comma>-separated or <blank>-separated, as specified in Guideline 8 of XBD [Section 12.2](#) (on page 215). The fields specified by *list* shall be written for all selected output lines. Fields selected by *list* that do not appear in the input shall be treated as empty output fields. (See the **-e** option.) Only specifically requested fields shall be written. The application shall ensure that *list* is a single command line argument.

join

Utilities

- 91901 **-t *char*** Use character *char* as a separator, for both input and output. Every appearance of
 91902 *char* in a line shall be significant. When this option is specified, the collating
 91903 sequence shall be the same as *sort* without the **-b** option.
- 91904 **-v *file_number*** Instead of the default output, produce a line only for each unpairable line in
 91905 *file_number*, where *file_number* is 1 or 2. If both **-v1** and **-v2** are specified, all
 91906 unpairable lines shall be output.
 91907
- 91908 **-1 *field*** Join on the *field*th field of file 1. Fields are decimal integers starting with 1.
 91909 **-2 *field*** Join on the *field*th field of file 2. Fields are decimal integers starting with 1.
- 91910 **OPERANDS**
 91911 The following operands shall be supported:
- 91912 *file1, file2* A pathname of a file to be joined. If either of the *file1* or *file2* operands is **'-'**, the
 91913 standard input shall be used in its place.
- 91914 **STDIN**
 91915 The standard input shall be used only if the *file1* or *file2* operand is **'-'**. See the INPUT FILES
 91916 section.
- 91917 **INPUT FILES**
 91918 The input files shall be text files.
- 91919 **ENVIRONMENT VARIABLES**
 91920 The following environment variables shall affect the execution of *join*:
- 91921 *LANG* Provide a default value for the internationalization variables that are unset or null.
 91922 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 91923 variables used to determine the values of locale categories.)
- 91924 *LC_ALL* If set to a non-empty string value, override the values of all the other
 91925 internationalization variables.
- 91926 *LC_COLLATE* Determine the locale of the collating sequence *join* expects to have been used when
 91927 the input files were sorted.
 91928
- 91929 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 91930 characters (for example, single-byte as opposed to multi-byte characters in
 91931 arguments and input files).
- 91932 *LC_MESSAGES* Determine the locale that should be used to affect the format and contents of
 91933 diagnostic messages written to standard error.
 91934
- 91935 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 91936 **ASYNCHRONOUS EVENTS**
 91937 Default.
- 91938 **STDOUT**
 91939 The *join* utility output shall be a concatenation of selected character fields. When the **-o** option
 91940 is not specified, the output shall be:
- 91941 "%s%s%s\n", <*join field*>, <*other file1 fields*>,
 91942 <*other file2 fields*>
 91943
- 91943 If the *join field* is not the first field in a file, the <*other file fields*> for that file shall be:

91944 <fields preceding join field>, <fields following join field>

91945 When the **-o** option is specified, the output format shall be:

91946 "%s\n", <concatenation of fields>

91947 where the concatenation of fields is described by the **-o** option, above.

91948 For either format, each field (except the last) shall be written with its trailing separator character.
 91949 If the separator is the default (<blank> characters), a single <space> shall be written after each
 91950 field (except the last).

91951 **STDERR**

91952 The standard error shall be used only for diagnostic messages.

91953 **OUTPUT FILES**

91954 None.

91955 **EXTENDED DESCRIPTION**

91956 None.

91957 **EXIT STATUS**

91958 The following exit values shall be returned:

91959 0 All input files were output successfully.

91960 >0 An error occurred.

91961 **CONSEQUENCES OF ERRORS**

91962 Default.

91963 **APPLICATION USAGE**

91964 Pathnames consisting of numeric digits or of the form *string.string* should not be specified
 91965 directly following the **-o** list.

91966 **EXAMPLES**

91967 The **-o 0** field essentially selects the union of the join fields. For example, given file **phone**:

!Name	Phone Number
Don	+1 123-456-7890
Hal	+1 234-567-8901
Yasushi	+2 345-678-9012

91972 and file **fax**:

!Name	Fax Number
Don	+1 123-456-7899
Keith	+1 456-789-0122
Yasushi	+2 345-678-9011

91977 (where the large expanses of white space are meant to each represent a single <tab>), the
 91978 command:

91979 `join -t "<tab>" -a 1 -a 2 -e '(unknown)' -o 0,1.2,2.2 phone fax`

91980 would produce:

!Name	Phone Number	Fax Number
Don	+1 123-456-7890	+1 123-456-7899
Hal	+1 234-567-8901	(unknown)
Keith	(unknown)	+1 456-789-0122
Yasushi	+2 345-678-9012	+2 345-678-9011

join*Utilities*

91986 Multiple instances of the same key will produce combinatorial results. The following:

91987 fa:

91988 a x

91989 a y

91990 a z

91991 fb:

91992 a p

91993 will produce:

91994 a x p

91995 a y p

91996 a z p

91997 And the following:

91998 fa:

91999 a b c

92000 a d e

92001 fb:

92002 a w x

92003 a y z

92004 a o p

92005 will produce:

92006 a b c w x

92007 a b c y z

92008 a b c o p

92009 a d e w x

92010 a d e y z

92011 a d e o p

92012 RATIONALE

92013 The **-e** option is only effective when used with **-o** because, unless specific fields are identified
 92014 using **-o**, *join* is not aware of what fields might be empty. The exception to this is the join field,
 92015 but identifying an empty join field with the **-e** string is not historical practice and some scripts
 92016 might break if this were changed.

92017 The 0 field in the **-o** list was adopted from the Tenth Edition version of *join* to satisfy
 92018 international objections that the *join* in the base documents does not support the “full join” or
 92019 “outer join” described in relational database literature. Although it has been possible to include
 92020 a join field in the output (by default, or by field number using **-o**), the join field could not be
 92021 included for an unpaired line selected by **-a**. The **-o 0** field essentially selects the union of the
 92022 join fields.

92023 This sort of outer join was not possible with the *join* commands in the base documents. The **-o 0**
 92024 field was chosen because it is an upwards-compatible change for applications. An alternative
 92025 was considered: have the join field represent the union of the fields in the files (where they are
 92026 identical for matched lines, and one or both are null for unmatched lines). This was not adopted
 92027 because it would break some historical applications.

92028 The ability to specify *file2* as **-** is not historical practice; it was added for completeness.

92029 The **-v** option is not historical practice, but was considered necessary because it permitted the
 92030 writing of *only* those lines that do not match on the join field, as opposed to the **-a** option, which
 92031 prints both lines that do and do not match. This additional facility is parallel with the **-v** option

- 92032 of *grep*.
- 92033 Some historical implementations have been encountered where a blank line in one of the input
92034 files was considered to be the end of the file; the description in this volume of POSIX.1-2008 does
92035 not cite this as an allowable case.
- 92036 Earlier versions of this standard allowed *-j*, *-j1*, *-j2* options, and a form of the *-o* option that
92037 allowed the *list* option-argument to be multiple arguments. These forms are no longer specified
92038 by POSIX.1-2008 but may be present in some implementations.
- 92039 **FUTURE DIRECTIONS**
- 92040 None.
- 92041 **SEE ALSO**
- 92042 *awk*, *comm*, *sort*, *uniq*
- 92043 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)
- 92044 **CHANGE HISTORY**
- 92045 First released in Issue 2.
- 92046 **Issue 6**
- 92047 The obsolescent *-j* options and the multi-argument *-o* option are removed in this version.
- 92048 The normative text is reworded to avoid use of the term “must” for application requirements.
- 92049 **Issue 7**
- 92050 Austin Group Interpretation 1003.1-2001 #027 is applied.
- 92051 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

kill

Utilities

92052 **NAME**92053 **kill** — terminate or signal processes92054 **SYNOPSIS**92055 **kill** *-s signal_name pid...*92056 **kill** *-l [exit_status]*92057 XSI **kill** [*-signal_name*] *pid...*92058 **kill** [*-signal_number*] *pid...*92059 **DESCRIPTION**92060 The *kill* utility shall send a signal to the process or processes specified by each *pid* operand.92061 For each *pid* operand, the *kill* utility shall perform actions equivalent to the *kill()* function
92062 defined in the System Interfaces volume of POSIX.1-2008 called with the following arguments:

- 92063 • The value of the *pid* operand shall be used as the *pid* argument.
- 92064 • The *sig* argument is the value specified by the *-s* option, *-signal_number* option, or the
- 92065 *-signal_name* option, or by SIGTERM, if none of these options is specified.

92066 **OPTIONS**

92067 XSI The *kill* utility shall conform to XBD Section 12.2 (on page 215), except that in the last two
 92068 SYNOPSIS forms, the *-signal_number* and *-signal_name* options are usually more than a single
 92069 character.

92070 The following options shall be supported:

92071 **-l** (The letter ell.) Write all values of *signal_name* supported by the implementation, if
 92072 no operand is given. If an *exit_status* operand is given and it is a value of the ' ? '
 92073 shell special parameter (see Section 2.5.2 (on page 2302) and *wait*) corresponding to
 92074 a process that was terminated by a signal, the *signal_name* corresponding to the
 92075 signal that terminated the process shall be written. If an *exit_status* operand is
 92076 given and it is the unsigned decimal integer value of a signal number, the
 92077 *signal_name* (the symbolic constant name without the **SIG** prefix defined in the
 92078 Base Definitions volume of POSIX.1-2008) corresponding to that signal shall be
 92079 written. Otherwise, the results are unspecified.

92080 **-s signal_name**
 92081 Specify the signal to send, using one of the symbolic names defined in the
 92082 **<signal.h>** header. Values of *signal_name* shall be recognized in a case-independent
 92083 fashion, without the **SIG** prefix. In addition, the symbolic name 0 shall be
 92084 recognized, representing the signal value zero. The corresponding signal shall be
 92085 sent instead of SIGTERM.

92086 XSI **-signal_name**
 92087 Equivalent to **-s signal_name**.

92088 XSI **-signal_number**
 92089 Specify a non-negative decimal integer, *signal_number*, representing the signal to be
 92090 used instead of SIGTERM, as the *sig* argument in the effective call to *kill()*. The
 92091 correspondence between integer values and the *sig* value used is shown in the
 92092 following list.

92093 The effects of specifying any *signal_number* other than those listed below are
 92094 undefined.

Utilities

kill

92095	0	0
92096	1	SIGHUP
92097	2	SIGINT
92098	3	SIGQUIT
92099	6	SIGABRT
92100	9	SIGKILL
92101	14	SIGALRM
92102	15	SIGTERM
92103	If the first argument is a negative integer, it shall be interpreted as a <i>−signal_number</i>	
92104	option, not as a negative <i>pid</i> operand specifying a process group.	

OPERANDS

The following operands shall be supported:

pid One of the following:

1. A decimal integer specifying a process or process group to be signaled. The process or processes selected by positive, negative, and zero values of the *pid* operand shall be as described for the *kill()* function. If process number 0 is specified, all processes in the current process group shall be signaled. For the effects of negative *pid* numbers, see the *kill()* function defined in the System Interfaces volume of POSIX.1-2008. If the first *pid* operand is negative, it should be preceded by "--" to keep it from being interpreted as an option.
2. A job control job ID (see XBD Section 3.203, on page 65) that identifies a background process group to be signaled. The job control job ID notation is applicable only for invocations of *kill* in the current shell execution environment; see Section 2.12 (on page 2331).

exit_status A decimal integer specifying a signal number or the exit status of a process terminated by a signal.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *kill*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

kill

Utilities

92136	<i>LC_MESSAGES</i>	
92137		Determine the locale that should be used to affect the format and contents of
92138		diagnostic messages written to standard error.
92139	XSI	<i>NLSPATH</i> Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
92140	ASYNCHRONOUS EVENTS	
92141		Default.
92142	STDOUT	
92143		When the <i>-l</i> option is not specified, the standard output shall not be used.
92144		When the <i>-l</i> option is specified, the symbolic name of each signal shall be written in the
92145		following format:
92146		<code>"%s%c", <signal_name>, <separator></code>
92147		where the <i><signal_name></i> is in uppercase, without the SIG prefix, and the <i><separator></i> shall be
92148		either a <i><newline></i> or a <i><space></i> . For the last signal written, <i><separator></i> shall be a <i><newline></i> .
92149		When both the <i>-l</i> option and <i>exit_status</i> operand are specified, the symbolic name of the
92150		corresponding signal shall be written in the following format:
92151		<code>"%s\n", <signal_name></code>
92152	STDERR	
92153		The standard error shall be used only for diagnostic messages.
92154	OUTPUT FILES	
92155		None.
92156	EXTENDED DESCRIPTION	
92157		None.
92158	EXIT STATUS	
92159		The following exit values shall be returned:
92160	0	At least one matching process was found for each <i>pid</i> operand, and the specified signal was
92161		successfully processed for at least one matching process.
92162	>0	An error occurred.
92163	CONSEQUENCES OF ERRORS	
92164		Default.
92165	APPLICATION USAGE	
92166		Process numbers can be found by using <i>ps</i> .
92167		The job control job ID notation is not required to work as expected when <i>kill</i> is operating in its
92168		own utility execution environment. In either of the following examples:
92169		<code>nohup kill %1 &</code>
92170		<code>system("kill %1");</code>
92171		the <i>kill</i> operates in a different environment and does not share the shell's understanding of job
92172		numbers.
92173	EXAMPLES	
92174		Any of the commands:
92175		<code>kill -9 100 -165</code>
92176		<code>kill -s kill 100 -165</code>

92177 `kill -s KILL 100 -165`

92178 sends the SIGKILL signal to the process whose process ID is 100 and to all processes whose
92179 process group ID is 165, assuming the sending process has permission to send that signal to the
92180 specified processes, and that they exist.

92181 The System Interfaces volume of POSIX.1-2008 and this volume of POSIX.1-2008 do not require
92182 specific signal numbers for any *signal_names*. Even the `-signal_number` option provides symbolic
92183 (although numeric) names for signals. If a process is terminated by a signal, its exit status
92184 indicates the signal that killed it, but the exact values are not specified. The `kill -l` option,
92185 however, can be used to map decimal signal numbers and exit status values into the name of a
92186 signal. The following example reports the status of a terminated job:

```
92187 job
92188 stat=$?
92189 if [ $stat -eq 0 ]
92190 then
92191     echo job completed successfully.
92192 elif [ $stat -gt 128 ]
92193 then
92194     echo job terminated by signal SIG$(kill -l $stat).
92195 else
92196     echo job terminated with error code $stat.
92197 fi
```

92198 To send the default signal to a process group (say 123), an application should use a command
92199 similar to one of the following:

```
92200 kill -TERM -123
92201 kill -- -123
```

92202 RATIONALE

92203 The `-l` option originated from the C shell, and is also implemented in the KornShell. The C shell
92204 output can consist of multiple output lines because the signal names do not always fit on a
92205 single line on some terminal screens. The KornShell output also included the implementation-
92206 defined signal numbers and was considered by the standard developers to be too difficult for
92207 scripts to parse conveniently. The specified output format is intended not only to accommodate
92208 the historical C shell output, but also to permit an entirely vertical or entirely horizontal listing
92209 on systems for which this is appropriate.

92210 An early proposal invented the name SIGNULL as a *signal_name* for signal 0 (used by the System
92211 Interfaces volume of POSIX.1-2008 to test for the existence of a process without sending it a
92212 signal). Since the *signal_name* 0 can be used in this case unambiguously, SIGNULL has been
92213 removed.

92214 An early proposal also required symbolic *signal_names* to be recognized with or without the **SIG**
92215 prefix. Historical versions of `kill` have not written the **SIG** prefix for the `-l` option and have not
92216 recognized the **SIG** prefix on *signal_names*. Since neither applications portability nor ease-of-use
92217 would be improved by requiring this extension, it is no longer required.

92218 To avoid an ambiguity of an initial negative number argument specifying either a signal number
92219 or a process group, POSIX.1-2008 mandates that it is always considered the former by
92220 implementations that support the XSI option. It also requires that conforming applications
92221 always use the `--` options terminator argument when specifying a process group, unless an
92222 option is also specified.

92223 The `-s` option was added in response to international interest in providing some form of `kill` that

kill

Utilities

92224 meets the Utility Syntax Guidelines.

92225 The job control job ID notation is not required to work as expected when *kill* is operating in its
 92226 own utility execution environment. In either of the following examples:

92227 `nohup kill %1 &`
 92228 `system("kill %1");`

92229 the *kill* operates in a different environment and does not understand how the shell has managed
 92230 its job numbers.

92231 **FUTURE DIRECTIONS**

92232 None.

92233 **SEE ALSO**

92234 [Chapter 2](#) (on page 2297), *ps*, *wait*

92235 XBD [Section 3.203](#) (on page 65), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215), [<signal.h>](#)

92236 XSH *kill()*

92237 **CHANGE HISTORY**

92238 First released in Issue 2.

92239 **Issue 6**

92240 The obsolescent versions of the SYNOPSIS are turned into non-obsolescent features of the XSI
 92241 option, corresponding to a similar change in the *trap* special built-in.

92242 **Issue 7**

92243 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

92244 **NAME**92245 lex — generate programs for lexical tasks (**DEVELOPMENT**)92246 **SYNOPSIS**92247 CD `lex [-t] [-n|-v] [file...]`92248 **DESCRIPTION**

92249 The *lex* utility shall generate C programs to be used in lexical processing of character input, and
 92250 that can be used as an interface to *yacc*. The C programs shall be generated from *lex* source code
 92251 and conform to the ISO C standard, without depending on any undefined, unspecified, or
 92252 implementation-defined behavior, except in cases where the code is copied directly from the
 92253 supplied source, or in cases that are documented by the implementation. Usually, the *lex* utility
 92254 shall write the program it generates to the file **lex.yy.c**; the state of this file is unspecified if *lex*
 92255 exits with a non-zero exit status. See the EXTENDED DESCRIPTION section for a complete
 92256 description of the *lex* input language.

92257 **OPTIONS**92258 The *lex* utility shall conform to XBD [Section 12.2](#) (on page 215), except for Guideline 9.

92259 The following options shall be supported:

- 92260 **-n** Suppress the summary of statistics usually written with the **-v** option. If no table
 92261 sizes are specified in the *lex* source code and the **-v** option is not specified, then **-n**
 92262 is implied.
- 92263 **-t** Write the resulting program to standard output instead of **lex.yy.c**.
- 92264 **-v** Write a summary of *lex* statistics to the standard output. (See the discussion of *lex*
 92265 table sizes in [Definitions in lex](#) (on page 2828).) If the **-t** option is specified and **-n**
 92266 is not specified, this report shall be written to standard error. If table sizes are
 92267 specified in the *lex* source code, and if the **-n** option is not specified, the **-v** option
 92268 may be enabled.

92269 **OPERANDS**

92270 The following operand shall be supported:

- 92271 *file* A pathname of an input file. If more than one such *file* is specified, all files shall be
 92272 concatenated to produce a single *lex* program. If no *file* operands are specified, or if
 92273 a *file* operand is **'-'**, the standard input shall be used.

92274 **STDIN**

92275 The standard input shall be used if no *file* operands are specified, or if a *file* operand is **'-'**. See
 92276 INPUT FILES.

92277 **INPUT FILES**

92278 The input files shall be text files containing *lex* source code, as described in the EXTENDED
 92279 DESCRIPTION section.

92280 **ENVIRONMENT VARIABLES**92281 The following environment variables shall affect the execution of *lex*:

- 92282 **LANG** Provide a default value for the internationalization variables that are unset or null.
 92283 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 92284 variables used to determine the values of locale categories.)
- 92285 **LC_ALL** If set to a non-empty string value, override the values of all the other
 92286 internationalization variables.

92287	<i>LC_COLLATE</i>	
92288		Determine the locale for the behavior of ranges, equivalence classes, and multi-
92289		character collating elements within regular expressions. If this variable is not set to
92290		the POSIX locale, the results are unspecified.
92291	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as
92292		characters (for example, single-byte as opposed to multi-byte characters in
92293		arguments and input files), and the behavior of character classes within regular
92294		expressions. If this variable is not set to the POSIX locale, the results are
92295		unspecified.
92296	<i>LC_MESSAGES</i>	
92297		Determine the locale that should be used to affect the format and contents of
92298		diagnostic messages written to standard error.
92299	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
92300	ASYNCHRONOUS EVENTS	
92301		Default.
92302	STDOUT	
92303		If the -t option is specified, the text file of C source code output of <i>lex</i> shall be written to
92304		standard output.
92305		If the -t option is not specified:
92306		• Implementation-defined informational, error, and warning messages concerning the
92307		contents of <i>lex</i> source code input shall be written to either the standard output or standard
92308		error.
92309		• If the -v option is specified and the -n option is not specified, <i>lex</i> statistics shall also be
92310		written to either the standard output or standard error, in an implementation-defined
92311		format. These statistics may also be generated if table sizes are specified with a ' % '
92312		operator in the <i>Definitions</i> section, as long as the -n option is not specified.
92313	STDERR	
92314		If the -t option is specified, implementation-defined informational, error, and warning messages
92315		concerning the contents of <i>lex</i> source code input shall be written to the standard error.
92316		If the -t option is not specified:
92317		1. Implementation-defined informational, error, and warning messages concerning the
92318		contents of <i>lex</i> source code input shall be written to either the standard output or
92319		standard error.
92320		2. If the -v option is specified and the -n option is not specified, <i>lex</i> statistics shall also be
92321		written to either the standard output or standard error, in an implementation-defined
92322		format. These statistics may also be generated if table sizes are specified with a ' % '
92323		operator in the <i>Definitions</i> section, as long as the -n option is not specified.
92324	OUTPUT FILES	
92325		A text file containing C source code shall be written to lex.yy.c , or to the standard output if the -t
92326		option is present.
92327	EXTENDED DESCRIPTION	
92328		Each input file shall contain <i>lex</i> source code, which is a table of regular expressions with
92329		corresponding actions in the form of C program fragments.
92330		When lex.yy.c is compiled and linked with the <i>lex</i> library (using the -ll operand with <i>c99</i>), the

resulting program shall read character input from the standard input and shall partition it into strings that match the given expressions.

When an expression is matched, these actions shall occur:

- The input string that was matched shall be left in *yytext* as a null-terminated string; *yytext* shall either be an external character array or a pointer to a character string. As explained in [Definitions in lex](#) (on page 2828), the type can be explicitly selected using the `%array` or `%pointer` declarations, but the default is implementation-defined.
- The external `int yyleng` shall be set to the length of the matching string.
- The expression's corresponding program fragment, or action, shall be executed.

During pattern matching, *lex* shall search the set of patterns for the single longest possible match. Among rules that match the same number of characters, the rule given first shall be chosen.

The general format of *lex* source shall be:

```
Definitions
%%
Rules
%%
UserSubroutines
```

The first `%%` is required to mark the beginning of the rules (regular expressions and actions); the second `%%` is required only if user subroutines follow.

Any line in the *Definitions* section beginning with a <blank> shall be assumed to be a C program fragment and shall be copied to the external definition area of the `lex.yy.c` file. Similarly, anything in the *Definitions* section included between delimiter lines containing only `"%{"` and `"%}"` shall also be copied unchanged to the external definition area of the `lex.yy.c` file.

Any such input (beginning with a <blank> or within `"%{"` and `"%}"` delimiter lines) appearing at the beginning of the *Rules* section before any rules are specified shall be written to `lex.yy.c` after the declarations of variables for the `yylex()` function and before the first line of code in `yylex()`. Thus, user variables local to `yylex()` can be declared here, as well as application code to execute upon entry to `yylex()`.

The action taken by *lex* when encountering any input beginning with a <blank> or within `"%{"` and `"%}"` delimiter lines appearing in the *Rules* section but coming after one or more rules is undefined. The presence of such input may result in an erroneous definition of the `yylex()` function.

C-language code in the input shall not contain C-language trigraphs. The C-language code within `"%{"` and `"%}"` delimiter lines shall not contain any lines consisting only of `"%}"`, or only of `%%`.

Definitions in lex

Definitions appear before the first "%" delimiter. Any line in this section not contained between "{ " and "}" lines and not beginning with a <blank> shall be assumed to define a *lex* substitution string. The format of these lines shall be:

name substitute

If a *name* does not meet the requirements for identifiers in the ISO C standard, the result is undefined. The string *substitute* shall replace the string {*name*} when it is used in a rule. The *name* string shall be recognized in this context only when the braces are provided and when it does not appear within a bracket expression or within double-quotes.

In the *Definitions* section, any line beginning with a <percent-sign> ('%') character and followed by an alphanumeric word beginning with either 's' or 'S' shall define a set of start conditions. Any line beginning with a '%' followed by a word beginning with either 'x' or 'X' shall define a set of exclusive start conditions. When the generated scanner is in a %s state, patterns with no state specified shall be also active; in a %x state, such patterns shall not be active. The rest of the line, after the first word, shall be considered to be one or more <blank>-separated names of start conditions. Start condition names shall be constructed in the same way as definition names. Start conditions can be used to restrict the matching of regular expressions to one or more states as described in [Regular Expressions in lex](#) (on page 2829).

Implementations shall accept either of the following two mutually-exclusive declarations in the *Definitions* section:

%array Declare the type of *yytext* to be a null-terminated character array.

%pointer Declare the type of *yytext* to be a pointer to a null-terminated character string.

The default type of *yytext* is implementation-defined. If an application refers to *yytext* outside of the scanner source file (that is, via an **extern**), the application shall include the appropriate **%array** or **%pointer** declaration in the scanner source file.

Implementations shall accept declarations in the *Definitions* section for setting certain internal table sizes. The declarations are shown in the following table.

Table 4-10 Table Size Declarations in *lex*

Declaration	Description	Minimum Value
%p n	Number of positions	2 500
%n n	Number of states	500
%a n	Number of transitions	2 000
%e n	Number of parse tree nodes	1 000
%k n	Number of packed character classes	1 000
%o n	Size of the output array	3 000

In the table, *n* represents a positive decimal integer, preceded by one or more <blank> characters. The exact meaning of these table size numbers is implementation-defined. The implementation shall document how these numbers affect the *lex* utility and how they are related to any output that may be generated by the implementation should limitations be encountered during the execution of *lex*. It shall be possible to determine from this output which of the table size values needs to be modified to permit *lex* to successfully generate tables for the input language. The values in the column Minimum Value represent the lowest values conforming implementations shall provide.

Rules in lex

The rules in *lex* source files are a table in which the left column contains regular expressions and the right column contains actions (C program fragments) to be executed when the expressions are recognized.

ERE action

ERE action

...

The extended regular expression (ERE) portion of a row shall be separated from *action* by one or more <blank> characters. A regular expression containing <blank> characters shall be recognized under one of the following conditions:

- The entire expression appears within double-quotes.
- The <blank> characters appear within double-quotes or square brackets.
- Each <blank> is preceded by a <backslash> character.

User Subroutines in lex

Anything in the user subroutines section shall be copied to *lex.yy.c* following *yylex()*.

Regular Expressions in lex

The *lex* utility shall support the set of extended regular expressions (see XBD [Section 9.4](#), on page 188), with the following additions and exceptions to the syntax:

"..." Any string enclosed in double-quotes shall represent the characters within the double-quotes as themselves, except that <backslash>-escapes (which appear in the following table) shall be recognized. Any <backslash>-escape sequence shall be terminated by the closing quote. For example, "\01"1" represents a single string: the octal value 1 followed by the character '1'.

<state>*r*, <state1, state2, ...>*r*

The regular expression *r* shall be matched only when the program is in one of the start conditions indicated by *state*, *state1*, and so on; see [Actions in lex](#) (on page 2831). (As an exception to the typographical conventions of the rest of this volume of POSIX.1-2008, in this case <state> does not represent a metavariable, but the literal angle-bracket characters surrounding a symbol.) The start condition shall be recognized as such only at the beginning of a regular expression.

r/x

The regular expression *r* shall be matched only if it is followed by an occurrence of regular expression *x* (*x* is the instance of trailing context, further defined below). The token returned in *yytext* shall only match *r*. If the trailing portion of *r* matches the beginning of *x*, the result is unspecified. The *r* expression cannot include further trailing context or the '\$' (match-end-of-line) operator; *x* cannot include the '^' (match-beginning-of-line) operator, nor trailing context, nor the '\$' operator. That is, only one occurrence of trailing context is allowed in a *lex* regular expression, and the '^' operator only can be used at the beginning of such an expression.

{*name*}

When *name* is one of the substitution symbols from the *Definitions* section, the string, including the enclosing braces, shall be replaced by the *substitute* value. The *substitute* value shall be treated in the extended regular expression as if it were enclosed in parentheses. No substitution shall occur if {*name*} occurs within a bracket expression or within double-quotes.

Within an ERE, a <backslash> character shall be considered to begin an escape sequence as specified in the table in XBD Chapter 5 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'). In addition, the escape sequences in the following table shall be recognized.

A literal <newline> cannot occur within an ERE; the escape sequence '\n' can be used to represent a <newline>. A <newline> shall not be matched by a period operator.

Table 4-11 Escape Sequences in *lex*

Escape Sequence	Description	Meaning
\digits	A <backslash> character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <backslash> for each byte.
\xdigits	A <backslash> character followed by the longest sequence of hexadecimal-digit characters (01234567abcdefABCDEF). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the hexadecimal integer.
\c	A <backslash> character followed by any character not described in this table or in the table in XBD Chapter 5 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').	The character 'c', unchanged.

Note: If a '\x' sequence needs to be immediately followed by a hexadecimal digit character, a sequence such as "\x1"1" can be used, which represents a character containing the value 1, followed by the character '1'.

The order of precedence given to extended regular expressions for *lex* differs from that specified in XBD Section 9.4 (on page 188). The order of precedence for *lex* shall be as shown in the following table, from high to low.

Note: The escaped characters entry is not meant to imply that these are operators, but they are included in the table to show their relationships to the true operators. The start condition, trailing context, and anchoring notations have been omitted from the table because of the placement restrictions described in this section; they can only appear at the beginning or ending of an ERE.

Table 4-12 ERE Precedence in *lex*

Extended Regular Expression	Precedence
collation-related bracket symbols	[=] [: :] [. .]
escaped characters	\<special character>
bracket expression	[]
quoting	" . . . "
grouping	()
definition	{ name }
single-character RE duplication	* + ?
concatenation	
interval expression	{ m , n }
alternation	

The ERE anchoring operators '*^*' and '*\$*' do not appear in the table. With *lex* regular expressions, these operators are restricted in their use: the '*^*' operator can only be used at the beginning of an entire regular expression, and the '*\$*' operator only at the end. The operators apply to the entire regular expression. Thus, for example, the pattern "*(^abc)|(def\$)*" is undefined; it can instead be written as two separate rules, one with the regular expression "*^abc*" and one with "*def\$*", which share a common action via the special '*|*' action (see below). If the pattern were written "*^abc|def\$*", it would match either "*abc*" or "*def*" on a line by itself.

Unlike the general ERE rules, embedded anchoring is not allowed by most historical *lex* implementations. An example of embedded anchoring would be for patterns such as "*(^|)foo(|\$)*" to match "*foo*" when it exists as a complete word. This functionality can be obtained using existing *lex* features:

```
^foo/[ \n]      |
" foo"/[ \n]    /* Found foo as a separate word. */
```

Note also that '*\$*' is a form of trailing context (it is equivalent to "*/\n*") and as such cannot be used with regular expressions containing another instance of the operator (see the preceding discussion of trailing context).

The additional regular expressions trailing-context operator '*/*' can be used as an ordinary character if presented within double-quotes, "*/"*"; preceded by a <backslash>, "*\"*"; or within a bracket expression, "*[/]*". The start-condition '*<*' and '*>*' operators shall be special only in a start condition at the beginning of a regular expression; elsewhere in the regular expression they shall be treated as ordinary characters.

Actions in *lex*

The action to be taken when an ERE is matched can be a C program fragment or the special actions described below; the program fragment can contain one or more C statements, and can also include special actions. The empty C statement '*;*' shall be a valid action; any string in the *lex.yy.c* input that matches the pattern portion of such a rule is effectively ignored or skipped. However, the absence of an action shall not be valid, and the action *lex* takes in such a condition is undefined.

The specification for an action, including C statements and special actions, can extend across several lines if enclosed in braces:

```
ERE <one or more blanks> { program statement
                           program statement }
```

- 92537 The program statements shall not contain unbalanced curly brace preprocessing tokens.
- 92538 The default action when a string in the input to a **lex.yy.c** program is not matched by any
 92539 expression shall be to copy the string to the output. Because the default behavior of a program
 92540 generated by *lex* is to read the input and copy it to the output, a minimal *lex* source program that
 92541 has just "%%" shall generate a C program that simply copies the input to the output unchanged.
- 92542 Four special actions shall be available:
- 92543 | ECHO; REJECT; BEGIN
- 92544 | The action ' | ' means that the action for the next rule is the action for this rule.
 92545 Unlike the other three actions, ' | ' cannot be enclosed in braces or be
 92546 <semicolon>-terminated; the application shall ensure that it is specified alone, with
 92547 no other actions.
- 92548 **ECHO;** Write the contents of the string *yytext* on the output.
- 92549 **REJECT;** Usually only a single expression is matched by a given string in the input.
 92550 **REJECT** means "continue to the next expression that matches the current input",
 92551 and shall cause whatever rule was the second choice after the current rule to be
 92552 executed for the same input. Thus, multiple rules can be matched and executed for
 92553 one input string or overlapping input strings. For example, given the regular
 92554 expressions "xyz" and "xy" and the input "xyz", usually only the regular
 92555 expression "xyz" would match. The next attempted match would start after **z**. If
 92556 the last action in the "xyz" rule is **REJECT**, both this rule and the "xy" rule
 92557 would be executed. The **REJECT** action may be implemented in such a fashion that
 92558 flow of control does not continue after it, as if it were equivalent to a **goto** to
 92559 another part of *yylex()*. The use of **REJECT** may result in somewhat larger and
 92560 slower scanners.
- 92561 **BEGIN** The action:
- 92562 BEGIN *newstate*;
- 92563 switches the state (start condition) to *newstate*. If the string *newstate* has not been
 92564 declared previously as a start condition in the *Definitions* section, the results are
 92565 unspecified. The initial state is indicated by the digit '0' or the token **INITIAL**.
- 92566 The functions or macros described below are accessible to user code included in the *lex* input. It
 92567 is unspecified whether they appear in the C code output of *lex*, or are accessible only through the
 92568 **-ll** operand to *c99* (the *lex* library).
- 92569 **int yylex(void)**
 92570 Performs lexical analysis on the input; this is the primary function generated by the *lex*
 92571 utility. The function shall return zero when the end of input is reached; otherwise, it shall
 92572 return non-zero values (tokens) determined by the actions that are selected.
- 92573 **int yymore(void)**
 92574 When called, indicates that when the next input string is recognized, it is to be appended to
 92575 the current value of *yytext* rather than replacing it; the value in *yylen* shall be adjusted
 92576 accordingly.
- 92577 **int yylless(int n)**
 92578 Retains *n* initial characters in *yytext*, NUL-terminated, and treats the remaining characters as
 92579 if they had not been read; the value in *yylen* shall be adjusted accordingly.

int input(void)

Returns the next character from the input, or zero on end-of-file. It shall obtain input from the stream pointer *yyin*, although possibly via an intermediate buffer. Thus, once scanning has begun, the effect of altering the value of *yyin* is undefined. The character read shall be removed from the input stream of the scanner without any processing by the scanner.

int unput(int c)

Returns the character 'c' to the input; *yytext* and *yytext* are undefined until the next expression is matched. The result of using *unput()* for more characters than have been input is unspecified.

The following functions shall appear only in the *lex* library accessible through the `-ll` operand; they can therefore be redefined by a conforming application:

int yywrap(void)

Called by *yylex()* at end-of-file; the default *yywrap()* shall always return 1. If the application requires *yylex()* to continue processing with another source of input, then the application can include a function *yywrap()*, which associates another file with the external variable `FILE * yyin` and shall return a value of zero.

int main(int argc, char *argv[])

Calls *yylex()* to perform lexical analysis, then exits. The user code can contain *main()* to perform application-specific operations, calling *yylex()* as applicable.

Except for *input()*, *unput()*, and *main()*, all external and static names generated by *lex* shall begin with the prefix **yy** or **YY**.

EXIT STATUS

The following exit values shall be returned:

- 0 Successful completion.
- >0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Conforming applications are warned that in the *Rules* section, an ERE without an action is not acceptable, but need not be detected as erroneous by *lex*. This may result in compilation or runtime errors.

The purpose of *input()* is to take characters off the input stream and discard them as far as the lexical analysis is concerned. A common use is to discard the body of a comment once the beginning of a comment is recognized.

The *lex* utility is not fully internationalized in its treatment of regular expressions in the *lex* source code or generated lexical analyzer. It would seem desirable to have the lexical analyzer interpret the regular expressions given in the *lex* source according to the environment specified when the lexical analyzer is executed, but this is not possible with the current *lex* technology. Furthermore, the very nature of the lexical analyzers produced by *lex* must be closely tied to the lexical requirements of the input language being described, which is frequently locale-specific anyway. (For example, writing an analyzer that is used for French text is not automatically useful for processing other languages.)

EXAMPLES

The following is an example of a *lex* program that implements a rudimentary scanner for a Pascal-like syntax:

```
%{
/* Need this for the call to atof() below. */
#include <math.h>
/* Need this for printf(), fopen(), and stdin below. */
#include <stdio.h>
}%

DIGIT      [0-9]
ID          [a-z][a-z0-9]*

%%

{DIGIT}+ {
    printf("An integer: %s (%d)\n", yytext,
        atoi(yytext));
}

{DIGIT}+"."{DIGIT}* {
    printf("A float: %s (%g)\n", yytext,
        atof(yytext));
}

if|then|begin|end|procedure|function {
    printf("A keyword: %s\n", yytext);
}

{ID}      printf("An identifier: %s\n", yytext);

"+"|"-"|"*"|"/"      printf("An operator: %s\n", yytext);

{"\[^\n]*"} /* Eat up one-line comments. */

[ \t\n]+ /* Eat up white space. */

. printf("Unrecognized character: %s\n", yytext);

%%

int main(int argc, char *argv[])
{
    ++argv, --argc; /* Skip over program name. */
    if (argc > 0)
        yyin = fopen(argv[0], "r");
    else
        yyin = stdin;

    yylex();
}
```

RATIONALE

Even though the *-c* option and references to the C language are retained in this description, *lex* may be generalized to other languages, as was done at one time for EFL, the Extended FORTRAN Language. Since the *lex* input specification is essentially language-independent, versions of this utility could be written to produce Ada, Modula-2, or Pascal code, and there are known historical implementations that do so.

The current description of *lex* bypasses the issue of dealing with internationalized EREs in the *lex* source code or generated lexical analyzer. If it follows the model used by *awk* (the source code is assumed to be presented in the POSIX locale, but input and output are in the locale specified by the environment variables), then the tables in the lexical analyzer produced by *lex* would interpret EREs specified in the *lex* source in terms of the environment variables specified when *lex* was executed. The desired effect would be to have the lexical analyzer interpret the EREs given in the *lex* source according to the environment specified when the lexical analyzer is executed, but this is not possible with the current *lex* technology.

The description of octal and hexadecimal-digit escape sequences agrees with the ISO C standard use of escape sequences.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this version.

There is no detailed output format specification. The observed behavior of *lex* under four different historical implementations was that none of these implementations consistently reported the line numbers for error and warning messages. Furthermore, there was a desire that *lex* be allowed to output additional diagnostic messages. Leaving message formats unspecified avoids these formatting questions and problems with internationalization.

Although the %x specifier for *exclusive* start conditions is not historical practice, it is believed to be a minor change to historical implementations and greatly enhances the usability of *lex* programs since it permits an application to obtain the expected functionality with fewer statements.

The %array and %pointer declarations were added as a compromise between historical systems. The System V-based *lex* copies the matched text to a *yytext* array. The *flex* program, supported in BSD and GNU systems, uses a pointer. In the latter case, significant performance improvements are available for some scanners. Most historical programs should require no change in porting from one system to another because the string being referenced is null-terminated in both cases. (The method used by *flex* in its case is to null-terminate the token in place by remembering the character that used to come right after the token and replacing it before continuing on to the next scan.) Multi-file programs with external references to *yytext* outside the scanner source file should continue to operate on their historical systems, but would require one of the new declarations to be considered strictly portable.

The description of EREs avoids unnecessary duplication of ERE details because their meanings within a *lex* ERE are the same as that for the ERE in this volume of POSIX.1-2008.

The reason for the undefined condition associated with text beginning with a <blank> or within "% { " and "% } " delimiter lines appearing in the *Rules* section is historical practice. Both the BSD and System V *lex* copy the indented (or enclosed) input in the *Rules* section (except at the beginning) to unreachable areas of the *yylex()* function (the code is written directly after a *break* statement). In some cases, the System V *lex* generates an error message or a syntax error, depending on the form of indented input.

The intention in breaking the list of functions into those that may appear in *lex.yy.c* versus those that only appear in *libl.a* is that only those functions in *libl.a* can be reliably redefined by a conforming application.

The descriptions of standard output and standard error are somewhat complicated because historical *lex* implementations chose to issue diagnostic messages to standard output (unless -t was given). POSIX.1-2008 allows this behavior, but leaves an opening for the more expected behavior of using standard error for diagnostics. Also, the System V behavior of writing the statistics when any table sizes are given is allowed, while BSD-derived systems can avoid it. The

- 92713 programmer can always precisely obtain the desired results by using either the `-t` or `-n` options.
- 92714 The OPERANDS section does not mention the use of `-` as a synonym for standard input; not all
- 92715 historical implementations support such usage for any of the *file* operands.
- 92716 A description of the *translation table* was deleted from early proposals because of its relatively
- 92717 low usage in historical applications.
- 92718 The change to the definition of the *input()* function that allows buffering of input presents the
- 92719 opportunity for major performance gains in some applications.
- 92720 The following examples clarify the differences between *lex* regular expressions and regular
- 92721 expressions appearing elsewhere in this volume of POSIX.1-2008. For regular expressions of the
- 92722 form `"r/x"`, the string matching *r* is always returned; confusion may arise when the beginning
- 92723 of *x* matches the trailing portion of *r*. For example, given the regular expression `"a*b/cc"` and
- 92724 the input `"aaabcc"`, *yytext* would contain the string `"aaab"` on this match. But given the
- 92725 regular expression `"x*/xy"` and the input `"xxxxy"`, the token `xxx`, not `xx`, is returned by some
- 92726 implementations because `xxx` matches `"x*"`.
- 92727 In the rule `"ab*/bc"`, the `"b*"` at the end of *r* extends *r*'s match into the beginning of the
- 92728 trailing context, so the result is unspecified. If this rule were `"ab/bc"`, however, the rule
- 92729 matches the text `"ab"` when it is followed by the text `"bc"`. In this latter case, the matching of *r*
- 92730 cannot extend into the beginning of *x*, so the result is specified.
- 92731 **FUTURE DIRECTIONS**
- 92732 None.
- 92733 **SEE ALSO**
- 92734 *c99*, *ed*, *yacc*
- 92735 XBD [Chapter 5](#) (on page 121), [Chapter 8](#) (on page 173), [Chapter 9](#) (on page 181), [Section 12.2](#) (on
- 92736 page 215)
- 92737 **CHANGE HISTORY**
- 92738 First released in Issue 2.
- 92739 **Issue 6**
- 92740 This utility is marked as part of the C-Language Development Utilities option.
- 92741 The obsolescent `-c` option is removed.
- 92742 The normative text is reworded to avoid use of the term "must" for application requirements.
- 92743 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/14 is applied, removing text describing
- 92744 behavior on systems with bytes consisting of more than eight bits.
- 92745 **Issue 7**
- 92746 Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the requirements for
- 92747 generated code to conform to the ISO C standard.
- 92748 Austin Group Interpretation 1003.1-2001 #191 is applied, clarifying the handling of C-language
- 92749 trigraphs and curly brace preprocessing tokens.
- 92750 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not
- 92751 apply.
- 92752 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Utilities

link

92753 **NAME**92754 link — call *link()* function92755 **SYNOPSIS**92756 XSI `link file1 file2`92757 **DESCRIPTION**92758 The *link* utility shall perform the function call:92759 `link(file1, file2);`92760 A user may need appropriate privileges to invoke the *link* utility.92761 **OPTIONS**

92762 None.

92763 **OPERANDS**

92764 The following operands shall be supported:

92765 *file1* The pathname of an existing file.92766 *file2* The pathname of the new directory entry to be created.92767 **STDIN**

92768 Not used.

92769 **INPUT FILES**

92770 Not used.

92771 **ENVIRONMENT VARIABLES**92772 The following environment variables shall affect the execution of *link*:92773 *LANG* Provide a default value for the internationalization variables that are unset or null.
92774 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
92775 variables used to determine the values of locale categories.)92776 *LC_ALL* If set to a non-empty string value, override the values of all the other
92777 internationalization variables.92778 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
92779 characters (for example, single-byte as opposed to multi-byte characters in
92780 arguments).92781 *LC_MESSAGES*
92782 Determine the locale that should be used to affect the format and contents of
92783 diagnostic messages written to standard error.92784 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.92785 **ASYNCHRONOUS EVENTS**

92786 Default.

92787 **STDOUT**

92788 None.

92789 **STDERR**

92790 The standard error shall be used only for diagnostic messages.

link*Utilities*92791 **OUTPUT FILES**

92792 None.

92793 **EXTENDED DESCRIPTION**

92794 None.

92795 **EXIT STATUS**

92796 The following exit values shall be returned:

92797 0 Successful completion.

92798 >0 An error occurred.

92799 **CONSEQUENCES OF ERRORS**

92800 Default.

92801 **APPLICATION USAGE**

92802 None.

92803 **EXAMPLES**

92804 None.

92805 **RATIONALE**

92806 None.

92807 **FUTURE DIRECTIONS**

92808 None.

92809 **SEE ALSO**92810 *ln*, *unlink*92811 XBD [Chapter 8](#) (on page 173)92812 XSH [link\(\)](#)92813 **CHANGE HISTORY**

92814 First released in Issue 5.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

ln — link files

SYNOPSIS

ln [-fs] [-L|-P] *source_file target_file*

ln [-fs] [-L|-P] *source_file... target_dir*

DESCRIPTION

In the first synopsis form, the *ln* utility shall create a new directory entry (link) at the destination path specified by the *target_file* operand. If the *-s* option is specified, a symbolic link shall be created for the file specified by the *source_file* operand. This first synopsis form shall be assumed when the final operand does not name an existing directory; if more than two operands are specified and the final is not an existing directory, an error shall result.

In the second synopsis form, the *ln* utility shall create a new directory entry (link), or if the *-s* option is specified a symbolic link, for each file specified by a *source_file* operand, at a destination path in the existing directory named by *target_dir*.

If the last operand specifies an existing file of a type not specified by the System Interfaces volume of POSIX.1-2008, the behavior is implementation-defined.

The corresponding destination path for each *source_file* shall be the concatenation of the target directory pathname, a <slash> character if the target directory pathname did not end in a <slash>, and the last pathname component of the *source_file*. The second synopsis form shall be assumed when the final operand names an existing directory.

For each *source_file*:

1. If the destination path exists and was created by a previous step, it is unspecified whether *ln* shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_files*; or will continue processing the current *source_file*. If the destination path exists:
 - a. If the *-f* option is not specified, *ln* shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_files*.
 - b. If *destination* names the same directory entry as the current *source_file* *ln* shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_files*.
 - c. Actions shall be performed equivalent to the *unlink()* function defined in the System Interfaces volume of POSIX.1-2008, called using *destination* as the *path* argument. If this fails for any reason, *ln* shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_files*.
2. If the *-s* option is specified, *ln* shall create a symbolic link named by the destination path and containing as its pathname *source_file*. The *ln* utility shall do nothing more with *source_file* and shall go on to any remaining files.
3. If *source_file* is a symbolic link:
 - a. If the *-P* option is in effect, actions shall be performed equivalent to the *linkat()* function with *source_file* as the *path1* argument, the destination path as the *path2* argument, *AT_FDCWD* as the *fd1* and *fd2* arguments, and zero as the *flag* argument.

ln*Utilities*

- 92859 b. If the **-L** option is in effect, actions shall be performed equivalent to the *linkat()*
 92860 function with *source_file* as the *path1* argument, the destination path as the *path2*
 92861 argument, **AT_FDCWD** as the *fd1* and *fd2* arguments, and
 92862 **AT_SYMLINK_FOLLOW** as the *flag* argument.

92863 The *ln* utility shall do nothing more with *source_file* and shall go on to any remaining files.

- 92864 4. Actions shall be performed equivalent to the *link()* function defined in the System
 92865 Interfaces volume of POSIX.1-2008 using *source_file* as the *path1* argument, and the
 92866 destination path as the *path2* argument.

OPTIONS

92867 The *ln* utility shall conform to XBD Section 12.2 (on page 215).

92868 The following options shall be supported:

- 92869 **-f** Force existing destination pathnames to be removed to allow the link.
- 92870 **-L** For each *source_file* operand that names a file of type symbolic link, create a (hard)
 92871 link to the file referenced by the symbolic link.
- 92872 **-P** For each *source_file* operand that names a file of type symbolic link, create a (hard)
 92873 link to the symbolic link itself.
- 92874 **-s** Create symbolic links instead of hard links. If the **-s** option is specified, the **-L** and
 92875 **-P** options shall be silently ignored.

92876 Specifying more than one of the mutually-exclusive options **-L** and **-P** shall not be considered
 92877 an error. The last option specified shall determine the behavior of the utility (unless the **-s**
 92878 option causes it to be ignored).

92879 If the **-s** option is not specified and neither a **-L** nor a **-P** option is specified, it is
 92880 implementation-defined which of the **-L** and **-P** options will be used as the default.

OPERANDS

92881 The following operands shall be supported:

- 92882 *source_file* A pathname of a file to be linked. If the **-s** option is specified, no restrictions on the
 92883 type of file or on its existence shall be made. If the **-s** option is not specified,
 92884 whether a directory can be linked is implementation-defined.
- 92885 *target_file* The pathname of the new directory entry to be created.
- 92886 *target_dir* A pathname of an existing directory in which the new directory entries are created.

STDIN

92887 Not used.

INPUT FILES

92888 None.

ENVIRONMENT VARIABLES

92889 The following environment variables shall affect the execution of *ln*:

- 92890 **LANG** Provide a default value for the internationalization variables that are unset or null.
 92891 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 92892 variables used to determine the values of locale categories.)
- 92893 **LC_ALL** If set to a non-empty string value, override the values of all the other
 92894 internationalization variables.

Utilities

ln

92900	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
92901		
92902		
92903	<i>LC_MESSAGES</i>	
92904		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
92905		
92906	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
92907	ASYNCHRONOUS EVENTS	
92908	Default.	
92909	STDOUT	
92910	Not used.	
92911	STDERR	
92912	The standard error shall be used only for diagnostic messages.	
92913	OUTPUT FILES	
92914	None.	
92915	EXTENDED DESCRIPTION	
92916	None.	
92917	EXIT STATUS	
92918	The following exit values shall be returned:	
92919	0 All the specified files were linked successfully.	
92920	>0 An error occurred.	
92921	CONSEQUENCES OF ERRORS	
92922	Default.	
92923	APPLICATION USAGE	
92924	None.	
92925	EXAMPLES	
92926	None.	
92927	RATIONALE	
92928	The CONSEQUENCES OF ERRORS section does not require <i>ln -f a b</i> to remove <i>b</i> if a subsequent link operation would fail.	
92929		
92930	Some historic versions of <i>ln</i> (including the one specified by the SVID) unlink the destination file, if it exists, by default. If the mode does not permit writing, these versions prompt for confirmation before attempting the unlink. In these versions the <i>-f</i> option causes <i>ln</i> not to attempt to prompt for confirmation.	
92931		
92932		
92933		
92934	This allows <i>ln</i> to succeed in creating links when the target file already exists, even if the file itself is not writable (although the directory must be). Early proposals specified this functionality.	
92935		
92936	This volume of POSIX.1-2008 does not allow the <i>ln</i> utility to unlink existing destination paths by default for the following reasons:	
92937		
92938	• The <i>ln</i> utility has historically been used to provide locking for shell applications, a usage that is incompatible with <i>ln</i> unlinking the destination path by default. There was no corresponding technical advantage to adding this functionality.	
92939		
92940		

- 92941 • This functionality gave *ln* the ability to destroy the link structure of files, which changes
- 92942 the historical behavior of *ln*.
- 92943 • This functionality is easily replicated with a combination of *rm* and *ln*.
- 92944 • It is not historical practice in many systems; BSD and BSD-derived systems do not support
- 92945 this behavior. Unfortunately, whichever behavior is selected can cause scripts written
- 92946 expecting the other behavior to fail.
- 92947 • It is preferable that *ln* perform in the same manner as the *link()* function, which does not
- 92948 permit the target to exist already.

92949 This volume of POSIX.1-2008 retains the *-f* option to provide support for shell scripts depending
 92950 on the SVID semantics. It seems likely that shell scripts would not be written to handle
 92951 prompting by *ln* and would therefore have specified the *-f* option.

92952 The *-f* option is an undocumented feature of many historical versions of the *ln* utility, allowing
 92953 linking to directories. These versions require modification.

92954 Early proposals of this volume of POSIX.1-2008 also required a *-i* option, which behaved like the
 92955 *-i* options in *cp* and *mv*, prompting for confirmation before unlinking existing files. This was not
 92956 historical practice for the *ln* utility and has been omitted.

92957 The *-L* and *-P* options allow for implementing both common behaviors of the *ln* utility. Earlier
 92958 versions of this standard did not specify these options and required the behavior now described
 92959 for the *-L* option. Many systems by default or as an alternative provided a non-conforming *ln*
 92960 utility with the behavior now described for the *-P* option. Since applications could not rely on *ln*
 92961 following links in practice, the *-L* and *-P* options were added to specify the desired behavior for
 92962 the application.

92963 The *-L* and *-P* options are ignored when *-s* is specified in order to allow an alias to be created to
 92964 alter the default behavior when creating hard links (for example, *alias ln='ln -L'*). They serve no
 92965 purpose when *-s* is specified, since *source_file* is then just a string to be used as the contents of
 92966 the created symbolic link and need not exist as a file.

92967 The specification ensures that *ln a a* with or without the *-f* option will not unlink the file *a*.
 92968 Earlier versions of this standard were unclear in this case.

92969 FUTURE DIRECTIONS

92970 None.

92971 SEE ALSO

92972 *chmod*, *find*, *pax*, *rm*

92973 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

92974 XSH *link()*, *unlink()*

92975 CHANGE HISTORY

92976 First released in Issue 2.

92977 Issue 6

92978 The *ln* utility is updated to include symbolic link processing as defined in the IEEE P1003.2b
 92979 draft standard.

92980 Issue 7

92981 Austin Group Interpretations 1003.1-2001 #164, #168, and #169 are applied.

92982 SD5-XCU-ERN-27 is applied, adding a new paragraph to the RATIONALE.

92983 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

*Utilities***ln**

92984
92985

The **-L** and **-P** options are added to make it implementation-defined whether the *ln* utility follows symbolic links.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

locale

Utilities

92986 **NAME**

92987 locale — get locale-specific information

92988 **SYNOPSIS**

92989 locale [-a|-m]

92990 locale [-ck] name...

92991 **DESCRIPTION**

92992 The *locale* utility shall write information about the current locale environment, or all public
 92993 locales, to the standard output. For the purposes of this section, a *public locale* is one provided by
 92994 the implementation that is accessible to the application.

92995 When *locale* is invoked without any arguments, it shall summarize the current locale
 92996 environment for each locale category as determined by the settings of the environment variables
 92997 defined in XBD Chapter 7 (on page 135).

92998 When invoked with operands, it shall write values that have been assigned to the keywords in
 92999 the locale categories, as follows:

- 93000 • Specifying a keyword name shall select the named keyword and the category containing
 93001 that keyword.
- 93002 • Specifying a category name shall select the named category and all keywords in that
 93003 category.

93004 **OPTIONS**93005 The *locale* utility shall conform to XBD Section 12.2 (on page 215).

93006 The following options shall be supported:

- 93007 **-a** Write information about all available public locales. The available locales shall
 93008 include **POSIX**, representing the POSIX locale. The manner in which the
 93009 implementation determines what other locales are available is implementation-
 93010 defined.
- 93011 **-c** Write the names of selected locale categories; see the STDOUT section. The **-c**
 93012 option increases readability when more than one category is selected (for example,
 93013 via more than one keyword name or via a category name). It is valid both with
 93014 and without the **-k** option.
- 93015 **-k** Write the names and values of selected keywords. The implementation may omit
 93016 values for some keywords; see the OPERANDS section.
- 93017 **-m** Write names of available charmaps; see XBD Section 6.1 (on page 125).

93018 **OPERANDS**

93019 The following operand shall be supported:

- 93020 **name** The name of a locale category as defined in XBD Chapter 7 (on page 135), the name
 93021 of a keyword in a locale category, or the reserved name **charmap**. The named
 93022 category or keyword shall be selected for output. If a single *name* represents both a
 93023 locale category name and a keyword name in the current locale, the results are
 93024 unspecified. Otherwise, both category and keyword names can be specified as
 93025 *name* operands, in any sequence. It is implementation-defined whether any
 93026 keyword values are written for the categories *LC_CTYPE* and *LC_COLLATE*.

Utilities

locale

93027 **STDIN**

93028 Not used.

93029 **INPUT FILES**

93030 None.

93031 **ENVIRONMENT VARIABLES**93032 The following environment variables shall affect the execution of *locale*:

93033 *LANG* Provide a default value for the internationalization variables that are unset or null.
 93034 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 93035 variables used to determine the values of locale categories.)

93036 *LC_ALL* If set to a non-empty string value, override the values of all the other
 93037 internationalization variables.

93038 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 93039 characters (for example, single-byte as opposed to multi-byte characters in
 93040 arguments and input files).

93041 *LC_MESSAGES*

93042 Determine the locale that should be used to affect the format and contents of
 93043 diagnostic messages written to standard error.

93044 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

93045 XSI The application shall ensure that the *LANG*, *LC_**, and *NLSPATH* environment variables specify
 93046 the current locale environment to be written out; they shall be used if the *-a* option is not
 93047 specified.

93048 **ASYNCHRONOUS EVENTS**

93049 Default.

93050 **STDOUT**93051 The *LANG* variable shall be written first using the format:

93052 "LANG=%s\n", <value>

93053 If *LANG* is not set or is an empty string, the value is the empty string.

93054 If *locale* is invoked without any options or operands, the names and values of the *LC_**
 93055 environment variables described in this volume of POSIX.1-2008 shall be written to the standard
 93056 output, one variable per line, and each line using the following format. Only those variables set
 93057 in the environment and not overridden by *LC_ALL* shall be written using this format:

93058 "%s=%s\n", <variable_name>, <value>

93059 The names of those *LC_** variables associated with locale categories defined in this volume of
 93060 POSIX.1-2008 that are not set in the environment or are overridden by *LC_ALL* shall be written
 93061 in the following format:

93062 "%s=\"%s\"\n", <variable_name>, <implied value>

93063 The <implied value> shall be the name of the locale that has been selected for that category by the
 93064 implementation, based on the values in *LANG* and *LC_ALL*, as described in XBD Chapter 8 (on
 93065 page 173).

93066 The <value> and <implied value> shown above shall be properly quoted for possible later reentry
 93067 to the shell. The <value> shall not be quoted using double-quotes (so that it can be distinguished
 93068 by the user from the <implied value> case, which always requires double-quotes).

locale

Utilities

93069 The *LC_ALL* variable shall be written last, using the first format shown above. If it is not set, it
 93070 shall be written as:

93071 "LC_ALL=\n"

93072 If any arguments are specified:

- 93073 1. If the *-a* option is specified, the names of all the public locales shall be written, each in the
 93074 following format:

93075 "%s\n", <locale name>

- 93076 2. If the *-c* option is specified, the names of all selected categories shall be written, each in
 93077 the following format:

93078 "%s\n", <category name>

93079 If keywords are also selected for writing (see following items), the category name output
 93080 shall precede the keyword output for that category.

93081 If the *-c* option is not specified, the names of the categories shall not be written; only the
 93082 keywords, as selected by the <name> operand, shall be written.

- 93083 3. If the *-k* option is specified, the names and values of selected keywords shall be written.
 93084 If a value is non-numeric and is not a compound keyword value, it shall be written in the
 93085 following format:

93086 "%s=\"%s\"\\n", <keyword name>, <keyword value>

93087 If a value is a non-numeric compound keyword value, it shall either be written in the
 93088 format:

93089 "%s=\"%s\"\\n", <keyword name>, <keyword value>

93090 where the <keyword value> is a single string of values separated by <semicolon>
 93091 characters, or it shall be written in the format:

93092 "%s=%s\\n", <keyword name>, <keyword value>

93093 where the <keyword value> is encoded as a set of strings, each enclosed in double-
 93094 quotation-marks, separated by <semicolon> characters.

93095 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the
 93096 *localedef -f* option when the locale was created shall be written, with the word **charmap** as
 93097 <keyword name>.

93098 If a value is numeric, it shall be written in one of the following formats:

93099 "%s=%d\\n", <keyword name>, <keyword value>

93100 "%s=%co\\n", <keyword name>, <escape character>, <keyword value>

93101 "%s=%cx%x\\n", <keyword name>, <escape character>, <keyword value>

93102 where the <escape character> is that identified by the **escape_char** keyword in the current
 93103 locale; see XBD Section 7.3 (on page 136).

93104 Compound keyword values (list entries) shall be separated in the output by <semicolon>
 93105 characters. When included in keyword values, the <semicolon>, <backslash>, double-
 93106 quote, and any control character shall be preceded (escaped) with the escape character.

4. If the **-k** option is not specified, selected keyword values shall be written, each in the following format:

```
"%s\n", <keyword value>
```

If the keyword was **charmap**, the name of the charmap (if any) that was specified via the *localedef* **-f** option when the locale was created shall be written.

5. If the **-m** option is specified, then a list of all available charmaps shall be written, each in the format:

```
"%s\n", <charmap>
```

where *<charmap>* is in a format suitable for use as the option-argument to the *localedef* **-f** option.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

- 0 All the requested information was found and output successfully.
- >0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

If the *LANG* environment variable is not set or set to an empty value, or one of the *LC_** environment variables is set to an unrecognized value, the actual locales assumed (if any) are implementation-defined as described in XBD Chapter 8 (on page 173).

Implementations are not required to write out the actual values for keywords in the categories *LC_CTYPE* and *LC_COLLATE*; however, they must write out the categories (allowing an application to determine, for example, which character classes are available).

EXAMPLES

In the following examples, the assumption is that locale environment variables are set as follows:

```
LANG=locale_x
LC_COLLATE=locale_y
```

The command *locale* would result in the following output:

```
LANG=locale_x
LC_CTYPE="locale_x"
LC_COLLATE=locale_y
LC_TIME="locale_x"
LC_NUMERIC="locale_x"
LC_MONETARY="locale_x"
LC_MESSAGES="locale_x"
LC_ALL=
```

93150 The order of presentation of the categories is not specified by this volume of POSIX.1-2008.

93151 The command:

93152 LC_ALL=POSIX locale -ck decimal_point

93153 would produce:

93154 LC_NUMERIC

93155 decimal_point="."

93156 The following command shows an application of *locale* to determine whether a user-supplied
93157 response is affirmative:

```
93158 if printf "%s\n" "$response" | grep -Eq "$(locale yesexpr)"
93159 then
93160     affirmative processing goes here
93161 else
93162     non-affirmative processing goes here
93163 fi
```

93164 RATIONALE

93165 The output for categories *LC_CTYPE* and *LC_COLLATE* has been made implementation-defined
93166 because there is a questionable value in having a shell script receive an entire array of characters.
93167 It is also difficult to return a logical collation description, short of returning a complete *localedef*
93168 source.

93169 The **-m** option was included to allow applications to query for the existence of charmaps. The
93170 output is a list of the charmaps (implementation-supplied and user-supplied, if any) on the
93171 system.

93172 The **-c** option was included for readability when more than one category is selected (for
93173 example, via more than one keyword name or via a category name). It is valid both with and
93174 without the **-k** option.

93175 The **charmap** keyword, which returns the name of the charmap (if any) that was used when the
93176 current locale was created, was included to allow applications needing the information to
93177 retrieve it.

93178 FUTURE DIRECTIONS

93179 None.

93180 SEE ALSO

93181 *localedef*

93182 XBD Section 6.1 (on page 125), Chapter 7 (on page 135), Chapter 8 (on page 173), Section 12.2 (on
93183 page 215)

93184 CHANGE HISTORY

93185 First released in Issue 4.

93186 Issue 5

93187 The FUTURE DIRECTIONS section is added.

93188 Issue 6

93189 The normative text is reworded to avoid use of the term “must” for application requirements.

93190 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/30 is applied, correcting an editorial error
93191 in the STDOUT section.

93192 **Issue 7**

93193 Austin Group Interpretations 1003.1-2001 #017, #021, and #088 are applied, clarifying the
93194 standard output for the `-k` option when `LANG` is not set or is an empty string.

93195 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

localedef

Utilities

93196 NAME

93197 localedef — define locale environment

93198 SYNOPSIS

93199 localedef [-c] [-f *charmap*] [-i *sourcefile*] [-u *code_set_name*] *name*

93200 DESCRIPTION

93201 The *localedef* utility shall convert source definitions for locale categories into a format usable by
 93202 the functions and utilities whose operational behavior is determined by the setting of the locale
 93203 environment variables defined in XBD Chapter 7 (on page 135). It is implementation-defined
 93204 whether users have the capability to create new locales, in addition to those supplied by the
 93205 implementation. If the symbolic constant POSIX2_LOCALEDEF is defined, the system supports
 93206 XSI the creation of new locales. On XSI-conformant systems, the symbolic constant
 93207 POSIX2_LOCALEDEF shall be defined.

93208 The utility shall read source definitions for one or more locale categories belonging to the same
 93209 locale from the file named in the *-i* option (if specified) or from standard input.

93210 The *name* operand identifies the target locale. The utility shall support the creation of *public*, or
 93211 generally accessible locales, as well as *private*, or restricted-access locales. Implementations may
 93212 restrict the capability to create or modify public locales to users with appropriate privileges.

93213 Each category source definition shall be identified by the corresponding environment variable
 93214 name and terminated by an **END** *category-name* statement. The following categories shall be
 93215 supported. In addition, the input may contain source for implementation-defined categories.

93216 *LC_CTYPE* Defines character classification and case conversion.

93217 *LC_COLLATE*
 93218 Defines collation rules.

93219 *LC_MONETARY*
 93220 Defines the format and symbols used in formatting of monetary information.

93221 *LC_NUMERIC*
 93222 Defines the decimal delimiter, grouping, and grouping symbol for non-monetary
 93223 numeric editing.

93224 *LC_TIME* Defines the format and content of date and time information.

93225 *LC_MESSAGES*
 93226 Defines the format and values of affirmative and negative responses.

93227 OPTIONS

93228 The *localedef* utility shall conform to XBD Section 12.2 (on page 215).

93229 The following options shall be supported:

93230 *-c* Create permanent output even if warning messages have been issued.

93231 *-f charmap* Specify the pathname of a file containing a mapping of character symbols and
 93232 collating element symbols to actual character encodings. The format of the
 93233 *charmap* is described in XBD Section 6.4 (on page 129). The application shall ensure
 93234 that this option is specified if symbolic names (other than collating symbols
 93235 defined in a **collating-symbol** keyword) are used. If the *-f* option is not present, an
 93236 implementation-defined character mapping shall be used.

93237 *-i inputfile* The pathname of a file containing the source definitions. If this option is not
 93238 present, source definitions shall be read from standard input. The format of the
 93239 *inputfile* is described in XBD Section 7.3 (on page 136).

93240 **-u** *code_set_name*

93241 Specify the name of a codeset used as the target mapping of character symbols and
 93242 collating element symbols whose encoding values are defined in terms of the
 93243 ISO/IEC 10646-1:2000 standard position constant values.

93244 **OPERANDS**

93245 The following operand shall be supported:

93246 *name* Identifies the locale; see XBD [Chapter 7](#) (on page 135) for a description of the use of
 93247 this name. If the name contains one or more <slash> characters, *name* shall be
 93248 interpreted as a pathname where the created locale definitions shall be stored. If
 93249 *name* does not contain any <slash> characters, the interpretation of the name is
 93250 implementation-defined and the locale shall be public. The ability to create public
 93251 locales in this way may be restricted to users with appropriate privileges. (As a
 93252 consequence of specifying one *name*, although several categories can be processed
 93253 in one execution, only categories belonging to the same locale can be processed.)

93254 **STDIN**

93255 Unless the **-i** option is specified, the standard input shall be a text file containing one or more
 93256 locale category source definitions, as described in XBD [Section 7.3](#) (on page 136). When lines are
 93257 continued using the escape character mechanism, there is no limit to the length of the
 93258 accumulated continued line.

93259 **INPUT FILES**

93260 The character set mapping file specified as the *charmap* option-argument is described in XBD
 93261 [Section 6.4](#) (on page 129). If a locale category source definition contains a **copy** statement, as
 93262 defined in XBD [Chapter 7](#) (on page 135), and the **copy** statement names a valid, existing locale,
 93263 then *localedef* shall behave as if the source definition had contained a valid category source
 93264 definition for the named locale.

93265 **ENVIRONMENT VARIABLES**

93266 The following environment variables shall affect the execution of *localedef*.

93267 *LANG* Provide a default value for the internationalization variables that are unset or null.
 93268 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 93269 variables used to determine the values of locale categories.)

93270 *LC_ALL* If set to a non-empty string value, override the values of all the other
 93271 internationalization variables.

93272 *LC_COLLATE* (This variable has no affect on *localedef*; the POSIX locale is used for this category.)

93274 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 93275 characters (for example, single-byte as opposed to multi-byte characters in
 93276 arguments and input files). This variable has no affect on the processing of *localedef*
 93277 input data; the POSIX locale is used for this purpose, regardless of the value of this
 93278 variable.

93279 *LC_MESSAGES*

93280 Determine the locale that should be used to affect the format and contents of
 93281 diagnostic messages written to standard error.

93282 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

localedef*Utilities*93283 **ASYNCHRONOUS EVENTS**

93284 Default.

93285 **STDOUT**

93286 The utility shall report all categories successfully processed, in an unspecified format.

93287 **STDERR**

93288 The standard error shall be used only for diagnostic messages.

93289 **OUTPUT FILES**93290 The format of the created output is unspecified. If the *name* operand does not contain a <slash>, the existence of an output file for the locale is unspecified.93292 **EXTENDED DESCRIPTION**

93293 When the **-u** option is used, the *code_set_name* option-argument shall be interpreted as an
 93294 implementation-defined name of a codeset to which the ISO/IEC 10646-1:2000 standard
 93295 position constant values shall be converted via an implementation-defined method. Both the
 93296 ISO/IEC 10646-1:2000 standard position constant values and other formats (decimal,
 93297 hexadecimal, or octal) shall be valid as encoding values within the *charmap* file. The codeset
 93298 represented by the implementation-defined name can be any codeset that is supported by the
 93299 implementation.

93300 When conflicts occur between the *charmap* specification of <*code_set_name*>, <*mb_cur_max*>, or
 93301 <*mb_cur_min*> and the implementation-defined interpretation of these respective items for the
 93302 codeset represented by the **-u** option-argument *code_set_name*, the result is unspecified.

93303 When conflicts occur between the *charmap* encoding values specified for symbolic names of
 93304 characters of the portable character set and the implementation-defined assignment of character
 93305 encoding values, the result is unspecified.

93306 If a non-printable character in the *charmap* has a width specified that is not **-1**, the result will be
 93307 undefined.

93308 **EXIT STATUS**

93309 The following exit values shall be returned:

- 93310 0 No errors occurred and the locales were successfully created.
- 93311 1 Warnings occurred and the locales were successfully created.
- 93312 2 The locale specification exceeded implementation limits or the coded character set or sets
 93313 used were not supported by the implementation, and no locale was created.
- 93314 3 The capability to create new locales is not supported by the implementation.
- 93315 >3 Warnings or errors occurred and no output was created.

93316 **CONSEQUENCES OF ERRORS**

93317 If an error is detected, no permanent output shall be created.

93318 If warnings occur, permanent output shall be created if the **-c** option was specified. The
 93319 following conditions shall cause warning messages to be issued:

- 93320 • If a symbolic name not found in the *charmap* file is used for the descriptions of the
 93321 *LC_CTYPE* or *LC_COLLATE* categories (for other categories, this shall be an error
 93322 condition).
- 93323 • If the number of operands to the **order** keyword exceeds the {*COLL_WEIGHTS_MAX*}
 93324 limit.

- 93325 • If optional keywords not supported by the implementation are present in the source.

93326 Other implementation-defined conditions may also cause warnings.

93327 APPLICATION USAGE

93328 The *charmap* definition is optional, and is contained outside the locale definition. This allows
 93329 both completely self-defined source files, and generic sources (applicable to more than one
 93330 codeset). To aid portability, all *charmap* definitions must use the same symbolic names for the
 93331 portable character set. As explained in XBD [Section 6.4](#) (on page 129), it is implementation-
 93332 defined whether or not users or applications can provide additional character set description
 93333 files. Therefore, the *-f* option might be operable only when an implementation-defined *charmap*
 93334 is named.

93335 EXAMPLES

93336 None.

93337 RATIONALE

93338 The output produced by the *localedef* utility is implementation-defined. The *name* operand is
 93339 used to identify the specific locale. (As a consequence, although several categories can be
 93340 processed in one execution, only categories belonging to the same locale can be processed.)

93341 FUTURE DIRECTIONS

93342 None.

93343 SEE ALSO

93344 [locale](#)

93345 XBD [Section 6.4](#) (on page 129), [Chapter 7](#) (on page 135), [Chapter 8](#) (on page 173), [Section 12.2](#) (on
 93346 page 215)

93347 CHANGE HISTORY

93348 First released in Issue 4.

93349 Issue 6

93350 The *-u* option is added, as specified in the IEEE P1003.2b draft standard.

93351 The normative text is reworded to avoid use of the term “must” for application requirements.

93352 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/15 is applied, rewording text in the
 93353 OPERANDS section describing the ability to create public locales.

93354 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/16 is applied, making the text consistent
 93355 with the descriptions of **WIDTH** and **WIDTH_DEFAULT** in the Base Definitions volume of
 93356 POSIX.1-2008.

93357 Issue 7

93358 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

logger

Utilities

93359 **NAME**93360 `logger` — log messages93361 **SYNOPSIS**93362 `logger string...`93363 **DESCRIPTION**

93364 The *logger* utility saves a message, in an unspecified manner and format, containing the *string*
 93365 operands provided by the user. The messages are expected to be evaluated later by personnel
 93366 performing system administration tasks.

93367 It is implementation-defined whether messages written in locales other than the POSIX locale
 93368 are effective.

93369 **OPTIONS**

93370 None.

93371 **OPERANDS**

93372 The following operand shall be supported:

93373 *string* One of the string arguments whose contents are concatenated together, in the order
 93374 specified, separated by single <space> characters.

93375 **STDIN**

93376 Not used.

93377 **INPUT FILES**

93378 None.

93379 **ENVIRONMENT VARIABLES**93380 The following environment variables shall affect the execution of *logger*:

93381 *LANG* Provide a default value for the internationalization variables that are unset or null.
 93382 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 93383 variables used to determine the values of locale categories.)

93384 *LC_ALL* If set to a non-empty string value, override the values of all the other
 93385 internationalization variables.

93386 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 93387 characters (for example, single-byte as opposed to multi-byte characters in
 93388 arguments).

93389 *LC_MESSAGES*

93390 Determine the locale that should be used to affect the format and contents of
 93391 diagnostic messages written to standard error. (This means diagnostics from *logger*
 93392 to the user or application, not diagnostic messages that the user is sending to the
 93393 system administrator.)

93394 *XSI NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

93395 **ASYNCHRONOUS EVENTS**

93396 Default.

93397 **STDOUT**

93398 Not used.

93399 **STDERR**

93400 The standard error shall be used only for diagnostic messages.

93401 **OUTPUT FILES**

93402 Unspecified.

93403 **EXTENDED DESCRIPTION**

93404 None.

93405 **EXIT STATUS**

93406 The following exit values shall be returned:

93407 0 Successful completion.

93408 >0 An error occurred.

93409 **CONSEQUENCES OF ERRORS**

93410 Default.

93411 **APPLICATION USAGE**

93412 This utility allows logging of information for later use by a system administrator or programmer
 93413 in determining why non-interactive utilities have failed. The locations of the saved messages,
 93414 their format, and retention period are all unspecified. There is no method for a conforming
 93415 application to read messages, once written.

93416 **EXAMPLES**

93417 A batch application, running non-interactively, tries to read a configuration file and fails; it may
 93418 attempt to notify the system administrator with:

93419 `logger myname: unable to read file foo. [timestamp]`

93420 **RATIONALE**

93421 The standard developers believed strongly that some method of alerting administrators to errors
 93422 was necessary. The obvious example is a batch utility, running non-interactively, that is unable to
 93423 read its configuration files or that is unable to create or write its results file. However, the
 93424 standard developers did not wish to define the format or delivery mechanisms as they have
 93425 historically been (and will probably continue to be) very system-specific, as well as involving
 93426 functionality clearly outside the scope of this volume of POSIX.1-2008.

93427 The text with *LC_MESSAGES* about diagnostic messages means diagnostics from *logger* to the
 93428 user or application, not diagnostic messages that the user is sending to the system administrator.

93429 Multiple *string* arguments are allowed, similar to *echo*, for ease-of-use.

93430 Like the utilities *mailx* and *lp*, *logger* is admittedly difficult to test. This was not deemed sufficient
 93431 justification to exclude these utilities from this volume of POSIX.1-2008. It is also arguable that
 93432 they are, in fact, testable, but that the tests themselves are not portable.

93433 **FUTURE DIRECTIONS**

93434 None.

93435 **SEE ALSO**

93436 *lp*, *mailx*, *write*

93437 XBD Chapter 8 (on page 173)

93438 **CHANGE HISTORY**

93439 First released in Issue 4.

logger

Utilities

93440 **Issue 7**
93441

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

logname

93442 **NAME**

93443 logname — return the user's login name

93444 **SYNOPSIS**

93445 logname

93446 **DESCRIPTION**

93447 The *logname* utility shall write the user's login name to standard output. The login name shall be
 93448 the string that would be returned by the *getlogin()* function defined in the System Interfaces
 93449 volume of POSIX.1-2008. Under the conditions where the *getlogin()* function would fail, the
 93450 *logname* utility shall write a diagnostic message to standard error and exit with a non-zero exit
 93451 status.

93452 **OPTIONS**

93453 None.

93454 **OPERANDS**

93455 None.

93456 **STDIN**

93457 Not used.

93458 **INPUT FILES**

93459 None.

93460 **ENVIRONMENT VARIABLES**

93461 The following environment variables shall affect the execution of *logname*:

93462 *LANG* Provide a default value for the internationalization variables that are unset or null.
 93463 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 93464 variables used to determine the values of locale categories.)

93465 *LC_ALL* If set to a non-empty string value, override the values of all the other
 93466 internationalization variables.

93467 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 93468 characters (for example, single-byte as opposed to multi-byte characters in
 93469 arguments).

93470 *LC_MESSAGES*
 93471 Determine the locale that should be used to affect the format and contents of
 93472 diagnostic messages written to standard error.

93473 *XSI NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

93474 **ASYNCHRONOUS EVENTS**

93475 Default.

93476 **STDOUT**

93477 The *logname* utility output shall be a single line consisting of the user's login name:

93478 "%s\n", <login name>

93479 **STDERR**

93480 The standard error shall be used only for diagnostic messages.

93481 **OUTPUT FILES**

93482 None.

logname*Utilities*93483 **EXTENDED DESCRIPTION**

93484 None.

93485 **EXIT STATUS**

93486 The following exit values shall be returned:

93487 0 Successful completion.

93488 >0 An error occurred.

93489 **CONSEQUENCES OF ERRORS**

93490 Default.

93491 **APPLICATION USAGE**93492 The *logname* utility explicitly ignores the *LOGNAME* environment variable because environment
93493 changes could produce erroneous results.93494 **EXAMPLES**

93495 None.

93496 **RATIONALE**93497 The **passwd** file is not listed as required because the implementation may have other means of
93498 mapping login names.93499 **FUTURE DIRECTIONS**

93500 None.

93501 **SEE ALSO**93502 *id*, *who*93503 XBD [Chapter 8](#) (on page 173)93504 XSH [getlogin\(\)](#)93505 **CHANGE HISTORY**

93506 First released in Issue 2.

Utilities

lp

NAME

lp — send files to a printer

SYNOPSIS

lp [-c] [-d *dest*] [-n *copies*] [-msw] [-o *option*]... [-t *title*] [*file*...]

DESCRIPTION

The *lp* utility shall copy the input files to an output destination in an unspecified manner. The default output destination should be to a hardcopy device, such as a printer or microfilm recorder, that produces non-volatile, human-readable documents. If such a device is not available to the application, or if the system provides no such device, the *lp* utility shall exit with a non-zero exit status.

The actual writing to the output device may occur some time after the *lp* utility successfully exits. During the portion of the writing that corresponds to each input file, the implementation shall guarantee exclusive access to the device.

The *lp* utility shall associate a unique *request ID* with each request.

Normally, a banner page is produced to separate and identify each print job. This page may be suppressed by implementation-defined conditions, such as an operator command or one of the *-o option* values.

OPTIONS

The *lp* utility shall conform to XBD Section 12.2 (on page 215).

The following options shall be supported:

- c** Exit only after further access to any of the input files is no longer required. The application can then safely delete or modify the files without affecting the output operation. Normally, files are not copied, but are linked whenever possible. If the *-c* option is not given, then the user should be careful not to remove any of the files before the request has been printed in its entirety. It should also be noted that in the absence of the *-c* option, any changes made to the named files after the request is made but before it is printed may be reflected in the printed output. On some implementations, *-c* may be on by default.
- d *dest*** Specify a string that names the destination (*dest*). If *dest* is a printer, the request shall be printed only on that specific printer. If *dest* is a class of printers, the request shall be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, and so on), requests for specific destinations need not be accepted. Destination names vary between systems.
If *-d* is not specified, and neither the *LPDEST* nor *PRINTER* environment variable is set, an unspecified destination is used. The *-d dest* option shall take precedence over *LPDEST*, which in turn shall take precedence over *PRINTER*. Results are undefined when *dest* contains a value that is not a valid destination name.
- m** Send mail (see *mailx*) after the files have been printed. By default, no mail is sent upon normal completion of the print request.
- n *copies*** Write *copies* number of copies of the files, where *copies* is a positive decimal integer. The methods for producing multiple copies and for arranging the multiple copies when multiple *file* operands are used are unspecified, except that each file shall be output as an integral whole, not interleaved with portions of other files.

lp*Utilities*

- 93551 **-o option** Specify printer-dependent or class-dependent *options*. Several such *options* may be
93552 collected by specifying the **-o** option more than once.
- 93553 **-s** Suppress messages from *lp*.
- 93554 **-t title** Write *title* on the banner page of the output.
- 93555 **-w** Write a message on the user's terminal after the files have been printed. If the user
93556 is not logged in, then mail shall be sent instead.

93557 **OPERANDS**

93558 The following operand shall be supported:

- 93559 *file* A pathname of a file to be output. If no *file* operands are specified, or if a *file*
93560 operand is ' - ', the standard input shall be used. If a *file* operand is used, but the
93561 **-c** option is not specified, the process performing the writing to the output device
93562 may have user and group permissions that differ from that of the process invoking
93563 *lp*.

93564 **STDIN**

93565 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is ' - '.

93566 See the INPUT FILES section.

93567 **INPUT FILES**

93568 The input files shall be text files.

93569 **ENVIRONMENT VARIABLES**93570 The following environment variables shall affect the execution of *lp*:

- 93571 *LANG* Provide a default value for the internationalization variables that are unset or null.
93572 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
93573 variables used to determine the values of locale categories.)
- 93574 *LC_ALL* If set to a non-empty string value, override the values of all the other
93575 internationalization variables.
- 93576 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
93577 characters (for example, single-byte as opposed to multi-byte characters in
93578 arguments and input files).
- 93579 *LC_MESSAGES* Determine the locale that should be used to affect the format and contents of
93580 diagnostic messages written to standard error and informative messages written to
93581 standard output.
- 93582 standard output.
- 93583 *LC_TIME* Determine the format and contents of date and time strings displayed in the *lp*
93584 banner page, if any.
- 93585 *LPDEST* Determine the destination. If the *LPDEST* environment variable is not set, the
93586 *PRINTER* environment variable shall be used. The **-d dest** option takes precedence
93587 over *LPDEST*. Results are undefined when **-d** is not specified and *LPDEST*
93588 contains a value that is not a valid destination name.
- 93589 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 93590 *PRINTER* Determine the output device or destination. If the *LPDEST* and *PRINTER*
93591 environment variables are not set, an unspecified output device is used. The **-d**
93592 *dest* option and the *LPDEST* environment variable shall take precedence over
93593 *PRINTER*. Results are undefined when **-d** is not specified, *LPDEST* is unset, and
93594 *PRINTER* contains a value that is not a valid device or destination name.

93595 TZ Determine the timezone used to calculate date and time strings displayed in the *lp*
 93596 banner page, if any. If *TZ* is unset or null, an unspecified default timezone shall be
 93597 used.

93598 ASYNCHRONOUS EVENTS

93599 Default.

93600 STDOUT

93601 The *lp* utility shall write a *request ID* to the standard output, unless *-s* is specified. The format of
 93602 the message is unspecified. The request ID can be used on systems supporting the historical
 93603 *cancel* and *lpstat* utilities.

93604 STDERR

93605 The standard error shall be used only for diagnostic messages.

93606 OUTPUT FILES

93607 None.

93608 EXTENDED DESCRIPTION

93609 None.

93610 EXIT STATUS

93611 The following exit values shall be returned:

93612 0 All input files were processed successfully.

93613 >0 No output device was available, or an error occurred.

93614 CONSEQUENCES OF ERRORS

93615 Default.

93616 APPLICATION USAGE

93617 The *pr* and *fold* utilities can be used to achieve reasonable formatting for the implementation's
 93618 default page size.

93619 A conforming application can use one of the *file* operands only with the *-c* option or if the file is
 93620 publicly readable and guaranteed to be available at the time of printing. This is because
 93621 POSIX.1-2008 gives the implementation the freedom to queue up the request for printing at
 93622 some later time by a different process that might not be able to access the file.

93623 EXAMPLES

93624 1. To print file *file*:

93625 `lp -c file`

93626 2. To print multiple files with headers:

93627 `pr file1 file2 | lp`

93628 RATIONALE

93629 The *lp* utility was designed to be a basic version of a utility that is already available in many
 93630 historical implementations. The standard developers considered that it should be implementable
 93631 simply as:

93632 `cat "$@" > /dev/lp`

93633 after appropriate processing of options, if that is how the implementation chose to do it and if
 93634 exclusive access could be granted (so that two users did not write to the device simultaneously).
 93635 Although in the future the standard developers may add other options to this utility, it should
 93636 always be able to execute with no options or operands and send the standard input to an

unspecified output device.

This volume of POSIX.1-2008 makes no representations concerning the format of the printed output, except that it must be “human-readable” and “non-volatile”. Thus, writing by default to a disk or tape drive or a display terminal would not qualify. (Such destinations are not prohibited when `-d dest`, `LPDEST`, or `PRINTER` are used, however.)

This volume of POSIX.1-2008 is worded such that a “print job” consisting of multiple input files, possibly in multiple copies, is guaranteed to print so that any one file is not intermixed with another, but there is no statement that all the files or copies have to print out together.

The `-c` option may imply a spooling operation, but this is not required. The utility can be implemented to wait until the printer is ready and then wait until it is finished. Because of that, there is no attempt to define a queuing mechanism (priorities, classes of output, and so on).

On some historical systems, the request ID reported on the `STDOUT` can be used to later cancel or find the status of a request using utilities not defined in this volume of POSIX.1-2008.

Although the historical System V `lp` and BSD `lpr` utilities have provided similar functionality, they used different names for the environment variable specifying the destination printer. Since the name of the utility here is `lp`, `LPDEST` (used by the System V `lp` utility) was given precedence over `PRINTER` (used by the BSD `lpr` utility). Since environments of users frequently contain one or the other environment variable, the `lp` utility is required to recognize both. If this was not done, many applications would send output to unexpected output devices when users moved from system to system.

Some have commented that `lp` has far too little functionality to make it worthwhile. Requests have proposed additional options or operands or both that added functionality. The requests included:

- Wording *requiring* the output to be “hardcopy”
- A requirement for multiple printers
- Options for supporting various page-description languages

Given that a compliant system is not required to even have a printer, placing further restrictions upon the behavior of the printer is not useful. Since hardcopy format is so application-dependent, it is difficult, if not impossible, to select a reasonable subset of functionality that should be required on all compliant systems.

The term *unspecified* is used in this section in lieu of *implementation-defined* as most known implementations would not be able to make definitive statements in their conformance documents; the existence and usage of printers is very dependent on how the system administrator configures each individual system.

Since the default destination, device type, queuing mechanisms, and acceptable forms of input are all unspecified, usage guidelines for what a conforming application can do are as follows:

- Use the command in a pipeline, or with `-c`, so that there are no permission problems and the files can be safely deleted or modified.
- Limit output to text files of reasonable line lengths and printable characters and include no device-specific formatting information, such as a page description language. The meaning of “reasonable” in this context can only be answered as a quality-of-implementation issue, but it should be apparent from historical usage patterns in the industry and the locale. The `pr` and `fold` utilities can be used to achieve reasonable formatting for the default page size of the implementation.

Alternatively, the application can arrange its installation in such a way that it requires the system administrator or operator to provide the appropriate information on *lp* options and environment variable values.

At a minimum, having this utility in this volume of POSIX.1-2008 tells the industry that conforming applications require a means to print output and provides at least a command name and *LPDEST* routing mechanism that can be used for discussions between vendors, application developers, and users. The use of “should” in the DESCRIPTION of *lp* clearly shows the intent of the standard developers, even if they cannot mandate that all systems (such as laptops) have printers.

This volume of POSIX.1-2008 does not specify what the ownership of the process performing the writing to the output device may be. If *-c* is not used, it is unspecified whether the process performing the writing to the output device has permission to read *file* if there are any restrictions in place on who may read *file* until after it is printed. Also, if *-c* is not used, the results of deleting *file* before it is printed are unspecified.

FUTURE DIRECTIONS

None.

SEE ALSO

mailx

XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the requirement to associate a unique request ID, and the normal generation of a banner page is added.
- In the OPTIONS section:
 - The *-d dest* description is expanded, but references to *lpstat* are removed.
 - The *-m*, *-o*, *-s*, *-t*, and *-w* options are added.
- In the ENVIRONMENT VARIABLES section, *LC_TIME* may now affect the execution.
- The STDOUT section is added.

The normative text is reworded to avoid use of the term “must” for application requirements.

The TZ entry is added to the ENVIRONMENT VARIABLES section.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

ls*Utilities*93716 **NAME**93717 **ls** — list directory contents93718 **SYNOPSIS**93719 XSI **ls** [**-ACFRS***acdfiklmnpqrstux1*] [**-H|-L**] [**-go**] [*file...*]93720 **DESCRIPTION**

93721 For each operand that names a file of a type other than directory or symbolic link to a directory,
 93722 *ls* shall write the name of the file as well as any requested, associated information. For each
 93723 operand that names a file of type directory, *ls* shall write the names of files contained within the
 93724 directory as well as any requested, associated information. Filenames beginning with a <period>
 93725 ('.') and any associated information shall not be written out unless explicitly referenced, the
 93726 **-A** or **-a** option is supplied, or an implementation-defined condition causes them to be written.
 93727 If one or more of the **-d**, **-F**, or **-l** options are specified, and neither the **-H** nor the **-L** option is
 93728 specified, for each operand that names a file of type symbolic link to a directory, *ls* shall write the
 93729 name of the file as well as any requested, associated information. If none of the **-d**, **-F**, or **-l**
 93730 options are specified, or the **-H** or **-L** options are specified, for each operand that names a file of
 93731 type symbolic link to a directory, *ls* shall write the names of files contained within the directory
 93732 as well as any requested, associated information. In each case where the names of files contained
 93733 within a directory are written, if the directory contains any symbolic links then *ls* shall evaluate
 93734 the file information and file type to be those of the symbolic link itself, unless the **-L** option is
 93735 specified.

93736 If no operands are specified, *ls* shall behave as if a single operand of dot ('.') had been
 93737 specified. If more than one operand is specified, *ls* shall write non-directory operands first; it
 93738 shall sort directory and non-directory operands separately according to the collating sequence in
 93739 the current locale.

93740 The *ls* utility shall detect infinite loops; that is, entering a previously visited directory that is an
 93741 ancestor of the last file encountered. When it detects an infinite loop, *ls* shall write a diagnostic
 93742 message to standard error and shall either recover its position in the hierarchy or terminate.

93743 **OPTIONS**93744 The *ls* utility shall conform to XBD [Section 12.2](#) (on page 215).

93745 The following options shall be supported:

- 93746 **-A** Write out all directory entries, including those whose names begin with a <period>
 93747 ('.') but excluding the entries dot and dot-dot (if they exist).
- 93748 **-C** Write multi-text-column output with entries sorted down the columns, according
 93749 to the collating sequence. The number of text columns and the column separator
 93750 characters are unspecified, but should be adapted to the nature of the output
 93751 device.
- 93752 **-F** Do not follow symbolic links named as operands unless the **-H** or **-L** options are
 93753 specified. Write a <slash> ('/') immediately after each pathname that is a
 93754 directory, an <asterisk> ('*') after each that is executable, a <vertical-line> ('|')
 93755 after each that is a FIFO, and an at-sign ('@') after each that is a symbolic link. For
 93756 other file types, other symbols may be written.
- 93757 **-H** Evaluate the file information and file type for symbolic links specified on the
 93758 command line to be those of the file referenced by the link, and not the link itself;
 93759 however, *ls* shall write the name of the link itself and not the file referenced by the
 93760 link.

Utilities

ls

93761		-L	Evaluate the file information and file type for all symbolic links (whether named on the command line or encountered in a file hierarchy) to be those of the file referenced by the link, and not the link itself; however, <i>ls</i> shall write the name of the link itself and not the file referenced by the link. When -L is used with -l , write the contents of symbolic links in the long format (see the STDOUT section).
93762			
93763			
93764			
93765			
93766		-R	Recursively list subdirectories encountered. When a symbolic link to a directory is encountered, the directory shall not be recursively listed unless the -L option is specified.
93767			
93768			
93769		-S	Sort with the primary key being file size (in decreasing order) and the secondary key being filename in the collating sequence (in increasing order).
93770			
93771		-a	Write out all directory entries, including those whose names begin with a <period> ('.').
93772			
93773		-c	Use time of last modification of the file status information (see <sys/stat.h> in the System Interfaces volume of POSIX.1-2008) instead of last modification of the file itself for sorting (-t) or writing (-l).
93774			
93775			
93776		-d	Do not follow symbolic links named as operands unless the -H or -L options are specified. Do not treat directories differently than other types of files. The use of -d with -R produces unspecified results.
93777			
93778			
93779		-f	List the entries in directory operands in the order they appear in the directory. The behavior for non-directory operands is unspecified. This option shall turn off -l , -t , -S , -s , and -r , and shall turn on -a .
93780			
93781			
93782	XSI	-g	The same as -l , except that the owner shall not be written.
93783		-i	For each file, write the file's file serial number (see <i>stat()</i> in the System Interfaces volume of POSIX.1-2008).
93784			
93785		-k	Set the block size for the -s option and the per-directory block count written for the -l , -n , -s , -g , and -o options (see the STDOUT section) to 1 024 bytes.
93786	XSI		
93787		-l	(The letter ell) Do not follow symbolic links named as operands unless the -H or -L options are specified. Write out in long format (see the STDOUT section). When -l (ell) is specified, -l (one) shall be assumed.
93788			
93789			
93790		-m	Stream output format; list files across the page, separated by <comma> characters.
93791		-n	The same as -l , except that the owner's UID and GID numbers shall be written, rather than the associated character strings.
93792			
93793	XSI	-o	The same as -l , except that the group shall not be written.
93794		-p	Write a <slash> ('/') after each filename if that file is a directory.
93795		-q	Force each instance of non-printable filename characters and <tab> characters to be written as the <question-mark> ('?') character. Implementations may provide this option by default if the output is to a terminal device.
93796			
93797			
93798		-r	Reverse the order of the sort to get reverse collating sequence oldest first, or smallest file size first depending on the other options given.
93799			
93800		-s	Indicate the total number of file system blocks consumed by each file displayed. If the -k option is also specified, the block size shall be 1 024 bytes; otherwise, the block size is implementation-defined.
93801			
93802			

ls*Utilities*

93803	-t	Sort with the primary key being time modified (most recently modified first) and the secondary key being filename in the collating sequence. For a symbolic link, the time used as the sort key is that of the symbolic link itself, unless <i>ls</i> is evaluating its file information to be that of the file referenced by the link (see the -H and -L options).
93804		
93805		
93806		
93807		
93808	-u	Use time of last access (see <sys/stat.h>) instead of last modification of the file for sorting (-t) or writing (-l).
93809		
93810	-x	The same as -C , except that the multi-text-column output is produced with entries sorted across, rather than down, the columns.
93811		
93812	-1	(The numeric digit one.) Force output to be one entry per line.
93813		Specifying more than one of the options in the following mutually-exclusive pairs shall not be
93814		considered an error: -C and -l (ell), -m and -l (ell), -x and -l (ell), -C and -1 (one), -H and -L ,
93815		-c and -u , -t and -S . The last option specified in each pair shall determine the output format.
93816	OPERANDS	
93817	The following operand shall be supported:	
93818	<i>file</i>	A pathname of a file to be written. If the file specified is not found, a diagnostic message shall be output on standard error.
93819		
93820	STDIN	
93821	Not used.	
93822	INPUT FILES	
93823	None.	
93824	ENVIRONMENT VARIABLES	
93825	The following environment variables shall affect the execution of <i>ls</i> :	
93826	COLUMNS	Determine the user's preferred column position width for writing multiple text-column output. If this variable contains a string representing a decimal integer, the <i>ls</i> utility shall calculate how many pathname text columns to write (see -C) based on the width provided. If COLUMNS is not set or invalid, an implementation-defined number of column positions shall be assumed, based on the implementation's knowledge of the output device. The column width chosen to write the names of files in any given directory shall be constant. Filenames shall not be truncated to fit into the multiple text-column output.
93827		
93828		
93829		
93830		
93831		
93832		
93833		
93834	LANG	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
93835		
93836		
93837	LC_ALL	If set to a non-empty string value, override the values of all the other internationalization variables.
93838		
93839	LC_COLLATE	
93840		Determine the locale for character collation information in determining the pathname collation sequence.
93841		
93842	LC_CTYPE	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments) and which characters are defined as printable (character class print).
93843		
93844		

93845		LC_MESSAGES	
93846			Determine the locale that should be used to affect the format and contents of
93847			diagnostic messages written to standard error.
93848		LC_TIME	Determine the format and contents for date and time strings written by <i>ls</i> .
93849	XSI	NLSPATH	Determine the location of message catalogs for the processing of LC_MESSAGES .
93850		TZ	Determine the timezone for date and time strings written by <i>ls</i> . If TZ is unset or
93851			null, an unspecified default timezone shall be used.
93852		ASYNCHRONOUS EVENTS	
93853			Default.
93854		STDOUT	
93855			The default format shall be to list one entry per line to standard output; the exceptions are to
93856			terminals or when one of the -C , -m , or -x options is specified. If the output is to a terminal, the
93857			format is implementation-defined.
93858			When -m is specified, the format used shall be:
93859			<code>"%s, %s, ...\\n", <filename1>, <filename2></code>
93860			where the largest number of filenames shall be written without exceeding the length of the line.
93861			If the -i option is specified, the file's file serial number (see <sys/stat.h>) shall be written in the
93862			following format before any other output for the corresponding entry:
93863			<code>%u ", <file serial number></code>
93864			If the -l option is specified without -L , the following information shall be written:
93865			<code>"%s %u %s %s %u %s %s\\n", <file mode>, <number of links>, <owner name>, <group name>, <number of bytes in the file>, <date and time>, <pathname></code>
93866			
93867			
93868			If the file is a symbolic link, this information shall be about the link itself and the <pathname>
93869			field shall be of the form:
93870			<code>"%s -> %s", <pathname of link>, <contents of link></code>
93871			If both -l and -L are specified, the following information shall be written:
93872			<code>"%s %u %s %s %u %s %s\\n", <file mode>, <number of links>, <owner name>, <group name>, <number of bytes in the file>, <date and time>, <pathname of link></code>
93873			
93874			
93875			where all fields except <pathname of link> shall be for the file resolved from the symbolic link.
93876	XSI		The -n , -g , and -o options use the same format as -l , but with omitted items and their
93877			associated <blank> characters. See the OPTIONS section.
93878			In both the preceding -l forms, if <owner name> or <group name> cannot be determined, or if -n
93879			is given, they shall be replaced with their associated numeric values using the format %u.
93880			The <date and time> field shall contain the appropriate date and timestamp of when the file was
93881			last modified. In the POSIX locale, the field shall be the equivalent of the output of the following
93882			<i>date</i> command:
93883			<code>date "+%b %e %H:%M"</code>
93884			if the file has been modified in the last six months, or:

93885 `date "+%b %e %Y"`

93886 (where two <space> characters are used between %e and %Y) if the file has not been modified in
 93887 the last six months or if the modification date is in the future, except that, in both cases, the final
 93888 <newline> produced by *date* shall not be included and the output shall be as if the *date*
 93889 command were executed at the time of the last modification date of the file rather than the
 93890 current time. When the *LC_TIME* locale category is not set to the POSIX locale, a different format
 93891 and order of presentation of this field may be used.

93892 If the file is a character special or block special file, the size of the file may be replaced with
 93893 implementation-defined information associated with the device in question.

93894 If the pathname was specified as a *file* operand, it shall be written as specified.

93895 XSI The file mode written under the `-l`, `-n`, `-g`, and `-o` options shall consist of the following format:

93896 `"%c%s%s%s", <entry type>, <owner permissions>,
 93897 <group permissions>, <other permissions>,
 93898 <optional alternate access method flag>`

93899 The <optional alternate access method flag> shall be the empty string if there is no alternate or
 93900 additional access control method associated with the file; otherwise, it shall be a string
 93901 containing a single printable character that is not a <blank>.

93902 The <entry type> character shall describe the type of file, as follows:

- 93903 d Directory.
- 93904 b Block special file.
- 93905 c Character special file.
- 93906 l (ell) Symbolic link.
- 93907 p FIFO.
- 93908 – Regular file.

93909 Implementations may add other characters to this list to represent other implementation-defined
 93910 file types.

93911 The next three fields shall be three characters each:

93912 <owner permissions>
 93913 Permissions for the file owner class (see XBD [Section 4.4](#), on page 108).

93914 <group permissions>
 93915 Permissions for the file group class.

93916 <other permissions>
 93917 Permissions for the file other class.

93918 Each field shall have three character positions:

- 93919 1. If 'r', the file is readable; if '–', the file is not readable.
- 93920 2. If 'w', the file is writable; if '–', the file is not writable.
- 93921 3. The first of the following that applies:

93922 S If in <owner permissions>, the file is not executable and set-user-ID mode is set. If in
 93923 <group permissions>, the file is not executable and set-group-ID mode is set.

Utilities

ls

93924	s	If in <owner permissions>, the file is executable and set-user-ID mode is set. If in
93925		<group permissions>, the file is executable and set-group-ID mode is set.
93926	XSI	T If in <other permissions> and the file is a directory, search permission is not granted to
93927		others, and the restricted deletion flag is set.
93928	XSI	t If in <other permissions> and the file is a directory, search permission is granted to
93929		others, and the restricted deletion flag is set.
93930	x	The file is executable or the directory is searchable.
93931	–	None of the attributes of 'S', 's', 'T', 't', or 'x' applies.
93932		Implementations may add other characters to this list for the third character position.
93933		Such additions shall, however, be written in lowercase if the file is executable or
93934		searchable, and in uppercase if it is not.
93935	XSI	If any of the –l , –n , –s , –g , or –o options is specified, each list of files within the directory shall be
93936		preceded by a status line indicating the number of file system blocks occupied by files in the
93937		directory in 512-byte units if the –k option is not specified, or 1 024-byte units if the –k option is
93938		specified, rounded up to the next integral number of units, if necessary. In the POSIX locale, the
93939		format shall be:
93940		"total %u\n", <number of units in the directory>
93941		If more than one directory, or a combination of non-directory files and directories are written,
93942		either as a result of specifying multiple operands, or the –R option, each list of files within a
93943		directory shall be preceded by:
93944		"\n%s:\n", <directory name>
93945		If this string is the first thing to be written, the first <newline> shall not be written. This output
93946		shall precede the number of units in the directory.
93947		If the –s option is given, each file shall be written with the number of blocks used by the file.
93948	XSI	Along with –C , –l , –m , or –x , the number and a <space> shall precede the filename; with –l , –n ,
93949		–g , or –o , they shall precede each line describing a file.
93950	STDERR	
93951		The standard error shall be used only for diagnostic messages.
93952	OUTPUT FILES	
93953		None.
93954	EXTENDED DESCRIPTION	
93955		None.
93956	EXIT STATUS	
93957		The following exit values shall be returned:
93958	0	Successful completion.
93959	>0	An error occurred.
93960	CONSEQUENCES OF ERRORS	
93961		Default.

APPLICATION USAGE

Many implementations use the <equals-sign> ('=') to denote sockets bound to the file system for the **-F** option. Similarly, many historical implementations use the 's' character to denote sockets as the entry type characters for the **-l** option.

It is difficult for an application to use every part of the file modes field of *ls -l* in a portable manner. Certain file types and executable bits are not guaranteed to be exactly as shown, as implementations may have extensions. Applications can use this field to pass directly to a user printout or prompt, but actions based on its contents should generally be deferred, instead, to the *test* utility.

The output of *ls* (with the **-l** and related options) contains information that logically could be used by utilities such as *chmod* and *touch* to restore files to a known state. However, this information is presented in a format that cannot be used directly by those utilities or be easily translated into a format that can be used. A character has been added to the end of the permissions string so that applications at least have an indication that they may be working in an area they do not understand instead of assuming that they can translate the permissions string into something that can be used. Future versions or related documents may define one or more specific characters to be used based on different standard additional or alternative access control mechanisms.

As with many of the utilities that deal with filenames, the output of *ls* for multiple files or in one of the long listing formats must be used carefully on systems where filenames can contain embedded white space. Systems and system administrators should institute policies and user training to limit the use of such filenames.

The number of disk blocks occupied by the file that it reports varies depending on underlying file system type, block size units reported, and the method of calculating the number of blocks. On some file system types, the number is the actual number of blocks occupied by the file (counting indirect blocks and ignoring holes in the file); on others it is calculated based on the file size (usually making an allowance for indirect blocks, but ignoring holes).

EXAMPLES

An example of a small directory tree being fully listed with *ls -laRF a* in the POSIX locale:

```
total 11
drwxr-xr-x  3 fox      prog          64 Jul  4 12:07 ./
drwxrwxrwx  4 fox      prog        3264 Jul  4 12:09 ../
drwxr-xr-x  2 fox      prog          48 Jul  4 12:07 b/
-rwxr--r-  1 fox      prog        572 Jul  4 12:07 foo*

a/b:
total 4
drwxr-xr-x  2 fox      prog          48 Jul  4 12:07 ./
drwxr-xr-x  3 fox      prog          64 Jul  4 12:07 ../
-rw-r--r-  1 fox      prog        700 Jul  4 12:07 bar
```

RATIONALE

Some historical implementations of the *ls* utility show all entries in a directory except dot and dot-dot when a superuser invokes *ls* without specifying the **-a** option. When "normal" users invoke *ls* without specifying **-a**, they should not see information about any files with names beginning with a <period> unless they were named as *file* operands.

Implementations are expected to traverse arbitrary depths when processing the **-R** option. The only limitation on depth should be based on running out of physical storage for keeping track of untraversed directories.

The **-1** (one) option was historically found in BSD and BSD-derived implementations only. It is required in this volume of POSIX.1-2008 so that conforming applications might ensure that output is one entry per line, even if the output is to a terminal.

The **-S** option was added in Issue 7, but had been provided by several implementations for many years. The description given in the standard documents historic practice, but does not match much of the documentation that described its behavior. Historical documentation typically described it as something like:

-S Sort by size (largest size first) instead of by name. Special character devices (listed last) are sorted by name.

even though the file type was never considered when sorting the output. Character special files do typically sort close to the end of the list because their file size on most implementations is zero. But they are sorted alphabetically with any other files that happen to have the same file size (zero), not sorted separately and added to the end.

Generally, this volume of POSIX.1-2008 is silent about what happens when options are given multiple times. In the cases of **-C**, **-l**, and **-1**, however, it does specify the results of these overlapping options. Since **ls** is one of the most aliased commands, it is important that the implementation perform intuitively. For example, if the alias were:

```
alias ls="ls -C"
```

and the user typed **ls -1**, single-text-column output should result, not an error.

Earlier versions of this standard did not describe the BSD **-A** option (like **-a**, but dot and dot-dot are not written out). It has been added due to widespread implementation.

Implementations may make **-q** the default for terminals to prevent trojan horse attacks on terminals with special escape sequences. This is not required because:

- Some control characters may be useful on some terminals; for example, a system might write them as `"\001"` or `"^A"`.
- Special behavior for terminals is not relevant to applications portability.

An early proposal specified that the *<optional alternate access method flag>* had to be **'+'** if there was an alternate access method used on the file or *<space>* if there was not. This was changed to be *<space>* if there is not and a single printable character if there is. This was done for three reasons:

1. There are historical implementations using characters other than **'+'**.
2. There are implementations that vary this character used in that position to distinguish between various alternate access methods in use.
3. The standard developers did not want to preclude future specifications that might need a way to specify more than one alternate access method.

Nonetheless, implementations providing a single alternate access method are encouraged to use **'+'**.

Earlier versions of this standard did not have the **-k** option, which meant that the **-s** option could not be used portably as its block size was implementation-defined, and the units used to specify the number of blocks occupied by files in a directory in an **ls -l** listing were fixed as 512-byte units. The **-k** option has been added to provide a way for the **-s** option to be used portably, and for consistency it also changes the aforementioned units from 512-byte to 1024-byte.

94052 The *<date and time>* field in the *-l* format is specified only for the POSIX locale. As noted, the
 94053 format can be different in other locales. No mechanism for defining this is present in this volume
 94054 of POSIX.1-2008, as the appropriate vehicle is a messaging system; that is, the format should be
 94055 specified as a “message”.

94056 FUTURE DIRECTIONS

94057 None.

94058 SEE ALSO

94059 *chmod*, *find*

94060 XBD Section 4.4 (on page 108), Chapter 8 (on page 173), Section 12.2 (on page 215), *<sys/stat.h>*

94061 XSH *fstatat()*

94062 CHANGE HISTORY

94063 First released in Issue 2.

94064 Issue 5

94065 A second FUTURE DIRECTION is added.

94066 Issue 6

94067 The following new requirements on POSIX implementations derive from alignment with the
 94068 Single UNIX Specification:

- 94069 • In the *-F* option, other symbols are allowed for other file types.

94070 Treatment of symbolic links is added, as defined in the IEEE P1003.2b draft standard.

94071 The Open Group Base Resolution bwg2001-010 is applied, adding the *T* and *t* fields as part of
 94072 the XSI option.

94073 Issue 7

94074 Austin Group Interpretation 1003.1-2001 #101 is applied, clarifying the optional alternate access
 94075 method flag in the STDOUT section.

94076 Austin Group Interpretation 1003.1-2001 #128 is applied, clarifying the DESCRIPTION and the
 94077 definition of the *-R* option.

94078 Austin Group Interpretation 1003.1-2001 #129 is applied, clarifying the behavior of *ls* when no
 94079 operands are specified.

94080 Austin Group Interpretation 1003.1-2001 #198 is applied, clarifying the requirements for the *-H*
 94081 option.

94082 SD5-XCU-ERN-50 is applied, adding the *-A* option.

94083 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

94084 The *-S* option is added from The Open Group Technical Standard, 2006, Extended API Set
 94085 Part 1.

94086 The *-f*, *-m*, *-n*, *-p*, *-s*, and *-x* options are moved from the XSI option to the Base.

94087 The description of the *-f*, *-s*, and *-t* options are revised and the *-k* option is added.

Utilities

m4

NAME

m4 — macro processor

SYNOPSIS

m4 [-s] [-D *name*[=*val*]]... [-U *name*]... *file*...

DESCRIPTION

The *m4* utility is a macro processor that shall read one or more text files, process them according to their included macro statements, and write the results to standard output.

OPTIONS

The *m4* utility shall conform to XBD Section 12.2 (on page 215), except that the order of the **-D** and **-U** options shall be significant, and options can be interspersed with operands.

The following options shall be supported:

-s Enable line synchronization output for the *c99* preprocessor phase (that is, **#line** directives).

-D *name*[=*val*]
Define *name* to *val* or to null if *=val* is omitted.

-U *name* Undefine *name*.

OPERANDS

The following operand shall be supported:

file A pathname of a text file to be processed. If no *file* is given, or if it is '-', the standard input shall be read.

STDIN

The standard input shall be a text file that is used if no *file* operand is given, or if it is '-'.

INPUT FILES

The input file named by the *file* operand shall be a text file.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *m4*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of **LC_MESSAGES**.

ASYNCHRONOUS EVENTS

Default.

94128 **STDOUT**

94129 The standard output shall be the same as the input files, after being processed for macro
 94130 expansion.

94131 **STDERR**

94132 The standard error shall be used to display strings with the **errprint** macro, macro tracing
 94133 enabled by the **traceon** macro, the defined text for macros written by the **dumpdef** macro, or for
 94134 diagnostic messages.

94135 **OUTPUT FILES**

94136 None.

94137 **EXTENDED DESCRIPTION**

94138 The *m4* utility shall compare each token from the input against the set of built-in and user-
 94139 defined macros. If the token matches the name of a macro, then the token shall be replaced by
 94140 the macro's defining text, if any, and rescanned for matching macro names. Once no portion of
 94141 the token matches the name of a macro, it shall be written to standard output. Macros may have
 94142 arguments, in which case the arguments shall be substituted into the defining text before it is
 94143 rescanned.

94144 Macro calls have the form:

94145 *name*(*arg1*, *arg2*, ..., *argn*)

94146 Macro names shall consist of letters, digits, and underscores, where the first character is not a
 94147 digit. Tokens not of this form shall not be treated as macros.

94148 The application shall ensure that the <left-parenthesis> immediately follows the name of the
 94149 macro. If a token matching the name of a macro is not followed by a <left-parenthesis>, it is
 94150 handled as a use of that macro without arguments.

94151 If a macro name is followed by a <left-parenthesis>, its arguments are the <comma>-separated
 94152 tokens between the <left-parenthesis> and the matching <right-parenthesis>. Unquoted white-
 94153 space characters preceding each argument shall be ignored. All other characters, including
 94154 trailing white-space characters, are retained. <comma> characters enclosed between <left-
 94155 parenthesis> and <right-parenthesis> characters do not delimit arguments.

94156 Arguments are positionally defined and referenced. The string "\$1" in the defining text shall be
 94157 replaced by the first argument. Systems shall support at least nine arguments; only the first nine
 94158 can be referenced, using the strings "\$1" to "\$9", inclusive. The string "\$0" is replaced with
 94159 the name of the macro. The string "\$#" is replaced by the number of arguments as a string. The
 94160 string "\$*" is replaced by a list of all of the arguments, separated by <comma> characters. The
 94161 string "\$@" is replaced by a list of all of the arguments separated by <comma> characters, and
 94162 each argument is quoted using the current left and right quoting strings. The string "\${"
 94163 produces unspecified behavior.

94164 If fewer arguments are supplied than are in the macro definition, the omitted arguments are
 94165 taken to be null. It is not an error if more arguments are supplied than are in the macro
 94166 definition.

94167 No special meaning is given to any characters enclosed between matching left and right quoting
 94168 strings, but the quoting strings are themselves discarded. By default, the left quoting string
 94169 consists of a grave accent (backquote) and the right quoting string consists of an acute accent
 94170 (single-quote); see also the **changequote** macro.

94171 Comments are written but not scanned for matching macro names; by default, the begin-
 94172 comment string consists of the <number-sign> character and the end-comment string consists of
 94173 a <newline>. See also the **changecom** and **dnl** macros.

The *m4* utility shall make available the following built-in macros. They can be redefined, but once this is done the original meaning is lost. Their values shall be null unless otherwise stated. In the descriptions below, the term *defining text* refers to the value of the macro: the second argument to the **define** macro, among other things. Except for the first argument to the **eval** macro, all numeric arguments to built-in macros shall be interpreted as decimal values. The string values produced as the defining text of the **decr**, **divnum**, **incr**, **index**, **len**, and **sysval** built-in macros shall be in the form of a decimal-constant as defined in the C language.

changecom The **changecom** macro shall set the begin-comment and end-comment strings. With no arguments, the comment mechanism shall be disabled. With a single non-null argument, that argument shall become the begin-comment and the `<newline>` shall become the end-comment string. With two non-null arguments, the first argument shall become the begin-comment string and the second argument shall become the end-comment string. The behavior is unspecified if either argument is provided but null. Systems shall support comment strings of at least five characters.

changequote The **changequote** macro shall set the begin-quote and end-quote strings. With no arguments, the quote strings shall be set to the default values (that is, `' '`). The behavior is unspecified if there is a single argument or either argument is null. With two non-null arguments, the first argument shall become the begin-quote string and the second argument shall become the end-quote string. Systems shall support quote strings of at least five characters.

decr The defining text of the **decr** macro shall be its first argument decremented by 1. It shall be an error to specify an argument containing any non-numeric characters. The behavior is unspecified if **decr** is not immediately followed by a `<left-parenthesis>`.

define The second argument shall become the defining text of the macro whose name is the first argument. It is unspecified whether the **define** macro deletes all prior definitions of the macro named by its first argument or preserves all but the current definition of the macro. The behavior is unspecified if **define** is not immediately followed by a `<left-parenthesis>`.

defn The defining text of the **defn** macro shall be the quoted definition (using the current quoting strings) of its arguments. The behavior is unspecified if **defn** is not immediately followed by a `<left-parenthesis>`.

divert The *m4* utility maintains nine temporary buffers, numbered 1 to 9, inclusive. When the last of the input has been processed, any output that has been placed in these buffers shall be written to standard output in buffer-numerical order. The **divert** macro shall divert future output to the buffer specified by its argument. Specifying no argument or an argument of 0 shall resume the normal output process. Output diverted to a stream with a negative number shall be discarded. Behavior is implementation-defined if a stream number larger than 9 is specified. It shall be an error to specify an argument containing any non-numeric characters.

divnum The defining text of the **divnum** macro shall be the number of the current output stream as a string.

dnl The **dnl** macro shall cause *m4* to discard all input characters up to and including the next `<newline>`.

94219	dumpdef	The dumpdef macro shall write the defined text to standard error for each of the macros specified as arguments, or, if no arguments are specified, for all macros.
94220		
94221	errprint	The errprint macro shall write its arguments to standard error. The behavior is unspecified if errprint is not immediately followed by a <left-parenthesis>.
94222		
94223	eval	The eval macro shall evaluate its first argument as an arithmetic expression, using signed integer arithmetic with at least 32-bit precision. At least the following C-language operators shall be supported, with precedence, associativity, and behavior as described in Section 1.1.2.1 (on page 2283):
94224		
94225		
94226		
94227		()
94228		unary +
94229		unary -
94230		~
94231		!
94232		binary *
94233		/
94234		%
94235		binary +
94236		binary -
94237		<<
94238		>>
94239		<
94240		<=
94241		>
94242		>=
94243		==
94244		!=
94245		binary &
94246		^
94247		
94248		&&
94249		
94250		Systems shall support octal and hexadecimal numbers as in the ISO C standard.
94251		The second argument, if specified, shall set the radix for the result; if the argument
94252		is blank or unspecified, the default is 10. Behavior is unspecified if the radix falls
94253		outside the range 2 to 36, inclusive. The third argument, if specified, sets the
94254		minimum number of digits in the result. Behavior is unspecified if the third
94255		argument is less than zero. It shall be an error to specify the second or third
94256		argument containing any non-numeric characters. The behavior is unspecified if
94257		eval is not immediately followed by a <left-parenthesis>.
94258	ifdef	If the first argument to the ifdef macro is defined, the defining text shall be the
94259		second argument. Otherwise, the defining text shall be the third argument, if
94260		specified, or the null string, if not. The behavior is unspecified if ifdef is not
94261		immediately followed by a <left-parenthesis>.
94262	ifelse	The ifelse macro takes three or more arguments. If the first two arguments
94263		compare as equal strings (after macro expansion of both arguments), the defining
94264		text shall be the third argument. If the first two arguments do not compare as equal
94265		strings and there are three arguments, the defining text shall be null. If the first two
94266		arguments do not compare as equal strings and there are four or five arguments,

94267		the defining text shall be the fourth argument. If the first two arguments do not
94268		compare as equal strings and there are six or more arguments, the first three
94269		arguments shall be discarded and processing shall restart with the remaining
94270		arguments. The behavior is unspecified if ifelse is not immediately followed by a
94271		<left-parenthesis>.
94272	include	The defining text for the include macro shall be the contents of the file named by
94273		the first argument. It shall be an error if the file cannot be read. The behavior is
94274		unspecified if include is not immediately followed by a <left-parenthesis>.
94275	incr	The defining text of the incr macro shall be its first argument incremented by 1. It
94276		shall be an error to specify an argument containing any non-numeric characters.
94277		The behavior is unspecified if incr is not immediately followed by a <left-
94278		parenthesis>.
94279	index	The defining text of the index macro shall be the first character position (as a
94280		string) in the first argument where a string matching the second argument begins
94281		(zero origin), or -1 if the second argument does not occur. The behavior is
94282		unspecified if index is not immediately followed by a <left-parenthesis>.
94283	len	The defining text of the len macro shall be the length (as a string) of the first
94284		argument. The behavior is unspecified if len is not immediately followed by a
94285		<left-parenthesis>.
94286	m4exit	Exit from the <i>m4</i> utility. If the first argument is specified, it is the exit code. The
94287		default is zero. It shall be an error to specify an argument containing any non-
94288		numeric characters.
94289	m4wrap	The first argument shall be processed when EOF is reached. If the m4wrap macro
94290		is used multiple times, the arguments specified shall be processed in the order in
94291		which the m4wrap macros were processed. The behavior is unspecified if m4wrap
94292		is not immediately followed by a <left-parenthesis>.
94293	OB maketemp	The defining text shall be the first argument, with any trailing 'X' characters
94294		replaced with the current process ID as a string. The behavior is unspecified if
94295		maketemp is not immediately followed by a <left-parenthesis>.
94296	mkstemp	The first argument shall be taken as a template for creating an empty file, with
94297		trailing 'X' characters replaced with characters from the portable filename
94298		character set. The behavior is unspecified if the first argument does not end in at
94299		least six 'X' characters. If a temporary file is successfully created, then the
94300		defining text of the macro shall be the name of the new file. The user ID of the file
94301		shall be set to the effective user ID of the process. The group ID of the file shall be
94302		set to the group ID of the file's parent directory or to the effective group ID of the
94303		process. The file access permission bits are set such that only the owner can both
94304		read and write the file, regardless of the current <i>umask</i> of the process. If a file could
94305		not be created, the defining text of the macro shall be the empty string. The
94306		behavior is unspecified if mkstemp is not immediately followed by a <left-
94307		parenthesis>.
94308	popdef	The popdef macro shall delete the current definition of its arguments, replacing
94309		that definition with the previous one. If there is no previous definition, the macro
94310		is undefined. The behavior is unspecified if popdef is not immediately followed by
94311		a <left-parenthesis>.

94312	pushdef	The pushdef macro shall be equivalent to the define macro with the exception that it shall preserve any current definition for future retrieval using the popdef macro.
94313		The behavior is unspecified if pushdef is not immediately followed by a <left-parenthesis>.
94314		
94315		
94316	shift	The defining text for the shift macro shall be all of its arguments except for the first one. The behavior is unspecified if shift is not immediately followed by a <left-parenthesis>.
94317		
94318		
94319	sinclude	The sinclude macro shall be equivalent to the include macro, except that it shall not be an error if the file is inaccessible. The behavior is unspecified if sinclude is not immediately followed by a <left-parenthesis>.
94320		
94321		
94322	substr	The defining text for the substr macro shall be the substring of the first argument beginning at the zero-offset character position specified by the second argument.
94323		The third argument, if specified, shall be the number of characters to select; if not specified, the characters from the starting point to the end of the first argument shall become the defining text. It shall not be an error to specify a starting point beyond the end of the first argument and the defining text shall be null. It shall be an error to specify an argument containing any non-numeric characters. The behavior is unspecified if substr is not immediately followed by a <left-parenthesis>.
94324		
94325		
94326		
94327		
94328		
94329		
94330		
94331	syscmd	The syscmd macro shall interpret its first argument as a shell command line. The defining text shall be the string result of that command. The string result shall not be rescanned for macros while setting the defining text. No output redirection shall be performed by the <i>m4</i> utility. The exit status value from the command can be retrieved using the sysval macro. The behavior is unspecified if syscmd is not immediately followed by a <left-parenthesis>.
94332		
94333		
94334		
94335		
94336		
94337	sysval	The defining text of the sysval macro shall be the exit value of the utility last invoked by the syscmd macro (as a string).
94338		
94339	traceon	The traceon macro shall enable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros. The trace output shall be written to standard error in an unspecified format.
94340		
94341		
94342	traceoff	The traceoff macro shall disable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros.
94343		
94344	translit	The defining text of the translit macro shall be the first argument with every character that occurs in the second argument replaced with the corresponding character from the third argument. The behavior is unspecified if the '-' character appears within the second or third argument anywhere besides the first or last character. The behavior is unspecified if translit is not immediately followed by a <left-parenthesis>.
94345		
94346		
94347		
94348		
94349		
94350	undefine	The undefine macro shall delete all definitions (including those preserved using the pushdef macro) of the macros named by its arguments. The behavior is unspecified if undefine is not immediately followed by a <left-parenthesis>.
94351		
94352		
94353	undivert	The undivert macro shall cause immediate output of any text in temporary buffers named as arguments, or all temporary buffers if no arguments are specified. Buffers can be undiverted into other temporary buffers. Undiverting shall discard the contents of the temporary buffer. The behavior is unspecified if an argument contains any non-numeric characters.
94354		
94355		
94356		
94357		

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred

If the **m4exit** macro is used, the exit value can be specified by the input file.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The **defn** macro is useful for renaming macros, especially built-ins.

Since **eval** defers to the ISO C standard, some operations have undefined behavior. In some implementations, division or remainder by zero cause a fatal signal, even if the division occurs on the short-circuited branch of "**&&**" or "**||**". Any operation that overflows in signed arithmetic produces undefined behavior. Likewise, using the **shift** operators with a shift amount that is not positive and smaller than the precision is undefined, as is shifting a negative number to the right. Historically, not all implementations obeyed C-language precedence rules: '**~**' and '**!**' were lower than '**==**'; '**==**' and '**!=**' were not lower than '**<**'; and '**|**' was not lower than '**^**'; the liberal use of "**()**" can force the desired precedence even with these non-compliant implementations. Furthermore, some traditional implementations treated '**^**' as an exponentiation operator, although most implementations now use "*******" as an extension for this purpose.

When a macro has been multiply defined via the **pushdef** macro, it is unspecified whether the **define** macro will alter only the most recent definition (as though by **popdef** and **pushdef**), or replace the entire stack of definitions with a single definition (as though by **undefine** and **pushdef**). An application desiring particular behavior for the **define** macro in this case can redefine it accordingly.

Applications should use the **mkstemp** macro instead of the obsolescent **maketemp** macro for creating temporary files.

EXAMPLES

If the file **m4src** contains the lines:

```
The value of 'VER' is "VER".
ifdef('VER', 'VER' is defined to be VER., VER is not defined.)
ifndef(VER, 1, 'VER' is 'VER'.)
ifndef(VER, 2, 'VER' is 'VER.', 'VER' is not 2.)
end
```

then the command

```
m4 m4src
```

or the command:

```
m4 -U VER m4src
```

produces the output:

```
The value of VER is "VER".
VER is not defined.

VER is not 2.
end
```

94401 The command:

94402 `m4 -D VER m4src`

94403 produces the output:

94404 The value of VER is "".

94405 VER is defined to be .

94406 VER is not 2.

94407 end

94408 The command:

94409 `m4 -D VER=1 m4src`

94410 produces the output:

94411 The value of VER is "1".

94412 VER is defined to be 1.

94413 VER is 1.

94414 VER is not 2.

94415 end

94416 The command:

94417 `m4 -D VER=2 m4src`

94418 produces the output:

94419 The value of VER is "2".

94420 VER is defined to be 2.

94421 VER is 2.

94422 end

94423 RATIONALE

94424 Historic System V-based behavior treated "\${" in a macro definition as two literal characters.
 94425 However, this sequence is left unspecified so that implementations may offer extensions such as
 94426 "\${\$11}" meaning the eleventh positional parameter. Macros can still be defined with
 94427 appropriate uses of nested quoting to result in a literal "\${" in the output after rescanning
 94428 removes the nested quotes.

94429 In the **translit** built-in, historic System V-based behavior treated '-' as a literal; GNU behavior
 94430 treats it as a range. This version of the standard allows either behavior.

94431 FUTURE DIRECTIONS

94432 None.

94433 SEE ALSO

94434 [c99](#)

94435 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

94436 CHANGE HISTORY

94437 First released in Issue 2.

94438 Issue 5

94439 The phrase "the defined text for macros written by the **dumpdef** macro" is added to the
 94440 description of **STDERR**, and the description of **dumpdef** is updated to indicate that output is
 94441 written to standard error. The description of **eval** is updated to indicate that the list of excluded
 94442 C operators excludes unary '&' and '.'. In the description of **ifdef**, the phrase "and it is not

94443 defined to be zero" is deleted.

94444 Issue 6

94445 In the EXTENDED DESCRIPTION, the **eval** text is updated to include a ' & ' character in the
94446 excepted list.

94447 The EXTENDED DESCRIPTION of **divert** is updated to clarify that there are only nine diversion
94448 buffers.

94449 The normative text is reworded to avoid use of the term "must" for application requirements.

94450 The Open Group Base Resolution bwg2000-006 is applied.

94451 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/31 is applied, replacing the EXAMPLES
94452 section.

94453 Issue 7

94454 Austin Group Interpretation 1003.1-2001 #117 is applied, marking the **maketemp** macro
94455 obsolescent and adding a new **mkstemp** macro.

94456 Austin Group Interpretation 1003.1-2001 #207 is applied, clarifying the handling of white-space
94457 characters that precede or trail any macro arguments.

94458 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not
94459 apply (options can be interspersed with operands).

94460 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

94461 SD5-XCU-ERN-99 is applied, clarifying the definition of the **divert** macro in the EXTENDED
94462 DESCRIPTION.

94463 SD5-XCU-ERN-100 is applied, clarifying the definition of the **syscmd** macro in the EXTENDED
94464 DESCRIPTION.

94465 SD5-XCU-ERN-101 is applied, clarifying the definition of the **undivert** macro in the EXTENDED
94466 DESCRIPTION.

94467 SD5-XCU-ERN-111 is applied to the EXTENDED DESCRIPTION, clarifying that the string "\$ { "
94468 produces unspecified behavior.

94469 SD5-XCU-ERN-112 is applied, updating the **changequote** macro.

94470 SD5-XCU-ERN-118 is applied, clarifying the definition of the **define** macro in the EXTENDED
94471 DESCRIPTION and APPLICATION USAGE sections.

94472 SD5-XCU-ERN-119 is applied, clarifying the definition of the **translit** macro in the EXTENDED
94473 DESCRIPTION and RATIONALE sections.

94474 SD5-XCU-ERN-130, SD5-XCU-ERN-131, and SD5-XCU-ERN-137 are applied.

94475 The **m4** utility is moved from the XSI option to the Base.

mailx

Utilities

94476 **NAME**

94477 mailx — process messages

94478 **SYNOPSIS**94479 **Send Mode**94480 mailx [-s *subject*] *address...*94481 **Receive Mode**

94482 UP mailx -e

94483 mailx [-HiNn] [-F] [-u *user*]94484 mailx -f [-HiNn] [-F] [*file*]94485 **DESCRIPTION**

94486 The *mailx* utility provides a message sending and receiving facility. It has two major modes,
 94487 selected by the options used: Send Mode and Receive Mode.

94488 On systems that do not support the User Portability Utilities option, an application using *mailx*
 94489 shall have the ability to send messages in an unspecified manner (Send Mode). Unless the first
 94490 character of one or more lines is <tilde> (' ~'), all characters in the input message shall appear in
 94491 the delivered message, but additional characters may be inserted in the message before it is
 94492 retrieved.

94493 UP On systems supporting the User Portability Utilities option, mail-receiving capabilities and other
 94494 interactive features, Receive Mode, described below, also shall be enabled.

94495 **Send Mode**

94496 Send Mode can be used by applications or users to send messages from the text in standard
 94497 input.

94498 UP **Receive Mode**

94499 Receive Mode is more oriented towards interactive users. Mail can be read and sent in this
 94500 interactive mode.

94501 When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to
 94502 messages. When sending mail, *mailx* allows editing, reviewing, and other modification of the
 94503 message as it is entered.

94504 Incoming mail shall be stored in one or more unspecified locations for each user, collectively
 94505 called the system *mailbox* for that user. When *mailx* is invoked in Receive Mode, the system
 94506 mailbox shall be the default place to find new mail. As messages are read, they shall be marked
 94507 to be moved to a secondary file for storage, unless specific action is taken. This secondary file is
 94508 called the **mbox** and is normally located in the directory referred to by the *HOME* environment
 94509 variable (see *MBOX* in the ENVIRONMENT VARIABLES section for a description of this file).
 94510 Messages shall remain in this file until explicitly removed. When the -f option is used to read
 94511 mail messages from secondary files, messages shall be retained in those files unless specifically
 94512 removed. All three of these locations—system mailbox, **mbox**, and secondary file—are referred
 94513 to in this section as simply “mailboxes”, unless more specific identification is required.

Utilities

mailx

OPTIONS

The *mailx* utility shall conform to XBD [Section 12.2](#) (on page 215).

The following options shall be supported. (Only the **-s subject** option shall be required on all systems. The other options are required only on systems supporting the User Portability Utilities option.)

-e Test for the presence of mail in the system mailbox. The *mailx* utility shall write nothing and exit with a successful return code if there is mail to read.

-f Read messages from the file named by the *file* operand instead of the system mailbox. (See also **folder**.) If no *file* operand is specified, read messages from **mbox** instead of the system mailbox.

-F Record the message in a file named after the first recipient. The name is the login-name portion of the address found first on the **To:** line in the mail header. Overrides the **record** variable, if set (see [Internal Variables in mailx](#), on page 2889).

-H Write a header summary only.

-i Ignore interrupts. (See also **ignore**.)

-n Do not initialize from the system default start-up file. See the EXTENDED DESCRIPTION section.

-N Do not write an initial header summary.

-s subject Set the **Subject** header field to *subject*. All characters in the *subject* string shall appear in the delivered message. The results are unspecified if *subject* is longer than {LINE_MAX} – 10 bytes or contains a <newline>.

-u user Read the system mailbox of the login name *user*. This shall only be successful if the invoking user has appropriate privileges to read the system mailbox of that user.

OPERANDS

The following operands shall be supported:

address Addressee of message. When **-n** is specified and no user start-up files are accessed (see the EXTENDED DESCRIPTION section), the user or application shall ensure this is an address to pass to the mail delivery system. Any system or user start-up files may enable aliases (see **alias** under [Commands in mailx](#), on page 2892) that may modify the form of *address* before it is passed to the mail delivery system.

file A pathname of a file to be read instead of the system mailbox when **-f** is specified. The meaning of the *file* option-argument shall be affected by the contents of the **folder** internal variable; see [Internal Variables in mailx](#) (on page 2889).

STDIN

When *mailx* is invoked in Send Mode (the first synopsis line), standard input shall be the message to be delivered to the specified addresses. When in Receive Mode, user commands shall be accepted from *stdin*. If the User Portability Utilities option is not supported, standard input lines beginning with a <tilde> ('~') character produce unspecified results.

If the User Portability Utilities option is supported, then in both Send and Receive Modes, standard input lines beginning with the escape character (usually <tilde> ('~')) shall affect processing as described in [Command Escapes in mailx](#) (on page 2901).

mailx*Utilities*94556 **INPUT FILES**

94557 When *mailx* is used as described by this volume of POSIX.1-2008, the *file* option-argument (see
 94558 the **-f** option) and the **mbox** shall be text files containing mail messages, formatted as described
 94559 in the OUTPUT FILES section. The nature of the system mailbox is unspecified; it need not be a
 94560 file.

94561 **ENVIRONMENT VARIABLES**

94562 UP Some of the functionality described in this section shall be provided on implementations that
 94563 support the User Portability Utilities option as described in the text, and is not further shaded
 94564 for this option.

94565 The following environment variables shall affect the execution of *mailx*:

94566 **DEAD** Determine the pathname of the file in which to save partial messages in case of
 94567 interrupts or delivery errors. The default shall be **dead.letter** in the directory
 94568 named by the *HOME* variable. The behavior of *mailx* in saving partial messages is
 94569 unspecified if the User Portability Utilities option is not supported and *DEAD* is
 94570 not defined with the value **/dev/null**.

94571 **EDITOR** Determine the name of a utility to invoke when the **edit** (see **Commands in mailx**,
 94572 on page 2892) or **~e** (see **Command Escapes in mailx**, on page 2901) command is
 94573 XSI used. The default editor is unspecified. On XSI-conformant systems it is *ed*. The
 94574 effects of this variable are unspecified if the User Portability Utilities option is not
 94575 supported.

94576 **HOME** Determine the pathname of the user's home directory.

94577 **LANG** Provide a default value for the internationalization variables that are unset or null.
 94578 (See XBD **Section 8.2** (on page 174) for the precedence of internationalization
 94579 variables used to determine the values of locale categories.)

94580 **LC_ALL** If set to a non-empty string value, override the values of all the other
 94581 internationalization variables.

94582 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 94583 characters (for example, single-byte as opposed to multi-byte characters in
 94584 arguments and input files) and the handling of case-insensitive address and
 94585 header-field comparisons.

94586 **LC_TIME** This variable may determine the format and contents of the date and time strings
 94587 written by *mailx*. This volume of POSIX.1-2008 specifies the effects of this variable
 94588 only for systems supporting the User Portability Utilities option.

94589 **LC_MESSAGES**

94590 Determine the locale that should be used to affect the format and contents of
 94591 diagnostic messages written to standard error and informative messages written to
 94592 standard output.

94593 **LISTER** Determine a string representing the command for writing the contents of the
 94594 **folder** directory to standard output when the **folders** command is given (see
 94595 **folders** in **Commands in mailx**, on page 2892). Any string acceptable as a
 94596 *command_string* operand to the *sh -c* command shall be valid. If this variable is null
 94597 or not set, the output command shall be *ls*. The effects of this variable are
 94598 unspecified if the User Portability Utilities option is not supported.

94599 **MAILRC** Determine the pathname of the start-up file. The default shall be **.mailrc** in the
 94600 directory referred to by the *HOME* environment variable. The behavior of *mailx* is
 94601 unspecified if the User Portability Utilities option is not supported and *MAILRC* is

Utilities

mailx

94602		not defined with the value <code>/dev/null</code> .
94603	<i>MBOX</i>	Determine a pathname of the file to save messages from the system mailbox that have been read. The exit command shall override this function, as shall saving the message explicitly in another file. The default shall be mbox in the directory named by the <i>HOME</i> variable. The effects of this variable are unspecified if the User Portability Utilities option is not supported.
94604		
94605		
94606		
94607		
94608	XSI	<i>NLSPATH</i> Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
94609	<i>PAGER</i>	Determine a string representing an output filtering or pagination command for writing the output to the terminal. Any string acceptable as a <i>command_string</i> operand to the <i>sh -c</i> command shall be valid. When standard output is a terminal device, the message output shall be piped through the command if the <i>mailx</i> internal variable crt is set to a value less the number of lines in the message; see Internal Variables in mailx (on page 2889). If the <i>PAGER</i> variable is null or not set, the paginator shall be either <i>more</i> or another paginator utility documented in the system documentation. The effects of this variable are unspecified if the User Portability Utilities option is not supported.
94610		
94611		
94612		
94613		
94614		
94615		
94616		
94617		
94618	<i>SHELL</i>	Determine the name of a preferred command interpreter. The default shall be <i>sh</i> . The effects of this variable are unspecified if the User Portability Utilities option is not supported.
94619		
94620		
94621	<i>TERM</i>	If the internal variable screen is not specified, determine the name of the terminal type to indicate in an unspecified manner the number of lines in a screenful of headers. If <i>TERM</i> is not set or is set to null, an unspecified default terminal type shall be used and the value of a screenful is unspecified. The effects of this variable are unspecified if the User Portability Utilities option is not supported.
94622		
94623		
94624		
94625		
94626	<i>TZ</i>	This variable may determine the timezone used to calculate date and time strings written by <i>mailx</i> . If <i>TZ</i> is unset or null, an unspecified default timezone shall be used.
94627		
94628		
94629	<i>VISUAL</i>	Determine a pathname of a utility to invoke when the visual command (see Commands in mailx , on page 2892) or ~v command-escape (see Command Escapes in mailx , on page 2901) is used. If this variable is null or not set, the full-screen editor shall be <i>vi</i> . The effects of this variable are unspecified if the User Portability Utilities option is not supported.
94630		
94631		
94632		
94633		
94634	ASYNCHRONOUS EVENTS	
94635		When <i>mailx</i> is in Send Mode and standard input is not a terminal, it shall take the standard action for all signals.
94636		
94637	UP	In Receive Mode, or in Send Mode when standard input is a terminal, if a SIGINT signal is received:
94638		
94639	UP	1. If in command mode, the current command, if there is one, shall be aborted, and a command-mode prompt shall be written.
94640		
94641		2. If in input mode:
94642	UP	a. If ignore is set, <i>mailx</i> shall write "@\n", discard the current input line, and continue processing, bypassing the message-abort mechanism described in item 2b.
94643		
94644		

mailx*Utilities*

- 94645 UP b. If the interrupt was received while sending mail, either when in Receive Mode or
 94646 in Send Mode, a message shall be written, and another subsequent interrupt, with
 94647 no other intervening characters typed, shall be required to abort the mail message.
 94648 UP If in Receive Mode and another interrupt is received, a command-mode prompt
 94649 shall be written. If in Send Mode and another interrupt is received, *mailx* shall
 94650 terminate with a non-zero status.

94651 In both cases listed in item b, if the message is not empty:

- 94652 UP i. If **save** is enabled and the file named by *DEAD* can be created, the message
 94653 shall be written to the file named by *DEAD*. If the file exists, the message
 94654 shall be written to replace the contents of the file.
- 94655 UP ii. If **save** is not enabled, or the file named by *DEAD* cannot be created, the
 94656 message shall not be saved.

94657 The *mailx* utility shall take the standard action for all other signals.

STDOUT

94659 In command and input modes, all output, including prompts and messages, shall be written to
 94660 standard output.

STDERR

94662 The standard error shall be used only for diagnostic messages.

OUTPUT FILES

94664 Various *mailx* commands and command escapes can create or add to files, including the **mbox**,
 94665 the dead-letter file, and secondary mailboxes. When *mailx* is used as described in this volume of
 94666 POSIX.1-2008, these files shall be text files, formatted as follows:

94667 line beginning with **From<space>**
 94668 [one or more *header-lines*; see [Commands in mailx](#) (on page 2892)]
 94669 *empty line*
 94670 [zero or more *body lines*
 94671 *empty line*]
 94672 [line beginning with **From<space>...]**

94673 where each message begins with the **From <space>** line shown, preceded by the beginning of
 94674 the file or an empty line. (The **From <space>** line is considered to be part of the message header,
 94675 but not one of the header-lines referred to in [Commands in mailx](#) (on page 2892); thus, it shall
 94676 not be affected by the **discard**, **ignore**, or **retain** commands.) The formats of the remainder of the
 94677 **From <space>** line and any additional header lines are unspecified, except that none shall be
 94678 empty. The format of a message body line is also unspecified, except that no line following an
 94679 empty line shall start with **From <space>**; *mailx* shall modify any such user-entered message
 94680 body lines (following an empty line and beginning with **From <space>**) by adding one or more
 94681 characters to precede the 'F'; it may add these characters to **From <space>** lines that are not
 94682 preceded by an empty line.

94683 When a message from the system mailbox or entered by the user is not a text file, it is
 94684 implementation-defined how such a message is stored in files written by *mailx*.

EXTENDED DESCRIPTION

94685 UP The functionality in the entire EXTENDED DESCRIPTION section shall be provided on
 94686 implementations supporting the User Portability Utilities option. The functionality described in
 94687 this section shall be provided on implementations that support the User Portability Utilities
 94688 option (and the rest of this section is not further shaded for this option).

The *mailx* utility need not support for all character encodings in all circumstances. For example, inter-system mail may be restricted to 7-bit data by the underlying network, 8-bit data need not be portable to non-internationalized systems, and so on. Under these circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used.

When *mailx* is invoked using one of the Receive Mode synopsis forms, it shall write a page of header-summary lines (if `-N` was not specified and there are messages, see below), followed by a prompt indicating that *mailx* can accept regular commands (see [Commands in mailx](#), on page 2892); this is termed *command mode*. The page of header-summary lines shall contain the first new message if there are new messages, or the first unread message if there are unread messages, or the first message. When *mailx* is invoked using the Send Mode synopsis and standard input is a terminal, if no subject is specified on the command line and the `asksub` variable is set, a prompt for the subject shall be written. At this point, *mailx* shall be in input mode. This input mode shall also be entered when using one of the Receive Mode synopsis forms and a reply or new message is composed using the **reply**, **Reply**, **followup**, **Followup**, or **mail** commands and standard input is a terminal. When the message is typed and the end of the message is encountered, the message shall be passed to the mail delivery software. Commands can be entered by beginning a line with the escape character (by default, `<tilde>` (' ~ ')) followed by a single command letter and optional arguments. See [Commands in mailx](#) (on page 2892) for a summary of these commands. It is unspecified what effect these commands will have if standard input is not a terminal when a message is entered using either the Send Mode synopsis, or the Read Mode commands **reply**, **Reply**, **followup**, **Followup**, or **mail**.

Note: For notational convenience, this section uses the default escape character, `<tilde>`, in all references and examples.

At any time, the behavior of *mailx* shall be governed by a set of environmental and internal variables. These are flags and valued parameters that can be set and cleared via the *mailx* **set** and **unset** commands.

Regular commands are of the form:

```
[command] [msglist] [argument ...]
```

If no *command* is specified in command mode, **next** shall be assumed. In input mode, commands shall be recognized by the escape character, and lines not treated as commands shall be taken as input for the message.

In command mode, each message shall be assigned a sequential number, starting with 1.

All messages have a state that shall affect how they are displayed in the header summary and how they are retained or deleted upon termination of *mailx*. There is at any time the notion of a *current* message, which shall be marked by a '`>`' at the beginning of a line in the header summary. When *mailx* is invoked using one of the Receive Mode synopsis forms, the current message shall be the first new message, if there is a new message, or the first unread message if there is an unread message, or the first message if there are any messages, or unspecified if there are no messages in the mailbox. Each command that takes an optional list of messages (*msglist*) or an optional single message (*message*) on which to operate shall leave the current message set to the highest-numbered message of the messages specified, unless the command deletes messages, in which case the current message shall be set to the first undeleted message (that is, a message not in the deleted state) after the highest-numbered message deleted by the command, if one exists, or the first undeleted message before the highest-numbered message deleted by the command, if one exists, or to an unspecified value if there are no remaining undeleted messages. All messages shall be in one of the following states:

mailx*Utilities*

94737	<i>new</i>	The message is present in the system mailbox and has not been viewed by the user or moved to any other state. Messages in state <i>new</i> when <i>mailx</i> quits shall be retained in the system mailbox.
94738		
94739		
94740	<i>unread</i>	The message has been present in the system mailbox for more than one invocation of <i>mailx</i> and has not been viewed by the user or moved to any other state. Messages in state <i>unread</i> when <i>mailx</i> quits shall be retained in the system mailbox.
94741		
94742		
94743	<i>read</i>	The message has been processed by one of the following commands: ~f, ~m, ~F, ~M, copy, mbox, next, pipe, print, Print, top, type, Type, undelete . The delete, dp, and dt commands may also cause the next message to be marked as <i>read</i> , depending on the value of the autoprint variable. Messages that are in the system mailbox and in state <i>read</i> when <i>mailx</i> quits shall be saved in the mbox , unless the internal variable hold was set. Messages that are in the mbox or in a secondary mailbox and in state <i>read</i> when <i>mailx</i> quits shall be retained in their current location.
94744		
94745		
94746		
94747		
94748		
94749		
94750		
94751	<i>deleted</i>	The message has been processed by one of the following commands: delete, dp, dt . Messages in state <i>deleted</i> when <i>mailx</i> quits shall be deleted. Deleted messages shall be ignored until <i>mailx</i> quits or changes mailboxes or they are specified to the undelete command; for example, the message specification <i>/string</i> shall only search the subject lines of messages that have not yet been deleted, unless the command operating on the list of messages is undelete . No deleted message or deleted message header shall be displayed by any <i>mailx</i> command other than undelete .
94752		
94753		
94754		
94755		
94756		
94757		
94758		
94759	<i>preserved</i>	The message has been processed by a preserve command. When <i>mailx</i> quits, the message shall be retained in its current location.
94760		
94761	<i>saved</i>	The message has been processed by one of the following commands: save or write . If the current mailbox is the system mailbox, and the internal variable keepsave is set, messages in the state <i>saved</i> shall be saved to the file designated by the MBOX variable (see the ENVIRONMENT VARIABLES section). If the current mailbox is the system mailbox, messages in the state <i>saved</i> shall be deleted from the current mailbox, when the quit or file command is used to exit the current mailbox.
94762		
94763		
94764		
94765		
94766		
94767		The header-summary line for each message shall indicate the state of the message.
94768		Many commands take an optional list of messages (<i>msglist</i>) on which to operate, which defaults to the current message. A <i>msglist</i> is a list of message specifications separated by <blank> characters, which can include:
94769		
94770		
94771	<i>n</i>	
94772	+	The next undeleted message, or the next deleted message for the undelete command.
94773	-	The next previous undeleted message, or the next previous deleted message for the undelete command.
94774		
94775	.	The current message.
94776	^	The first undeleted message, or the first deleted message for the undelete command.
94777	\$	The last message.
94778	*	All messages.

Utilities

mailx

94779	<i>n-m</i>	An inclusive range of message numbers.
94780	<i>address</i>	All messages from <i>address</i> ; any address as shown in a header summary shall be matchable in this form.
94781		
94782	<i>/string</i>	All messages with <i>string</i> in the subject line (case ignored).
94783	<i>:c</i>	All messages of type <i>c</i> , where <i>c</i> shall be one of:
94784	<i>d</i>	Deleted messages.
94785	<i>n</i>	New messages.
94786	<i>o</i>	Old messages (any not in state <i>read</i> or <i>new</i>).
94787	<i>r</i>	Read messages.
94788	<i>u</i>	Unread messages.

94789 Other commands take an optional message (*message*) on which to operate, which defaults to the
 94790 current message. All of the forms allowed for *msglist* are also allowed for *message*, but if more
 94791 than one message is specified, only the first shall be operated on.

94792 Other arguments are usually arbitrary strings whose usage depends on the command involved.

94793 Start-Up in mailx

94794 At start-up time, *mailx* shall take the following steps in sequence:

- 94795 1. Establish all variables at their stated default values.
- 94796 2. Process command line options, overriding corresponding default values.
- 94797 3. Import any of the *DEAD*, *EDITOR*, *MBOX*, *LISTER*, *PAGER*, *SHELL*, or *VISUAL* variables
 94798 that are present in the environment, overriding the corresponding default values.
- 94799 4. Read *mailx* commands from an unspecified system start-up file, unless the *-n* option is
 94800 given, to initialize any internal *mailx* variables and aliases.
- 94801 5. Process the start-up file of *mailx* commands named in the user *MAILRC* variable.

94802 Most regular *mailx* commands are valid inside start-up files, the most common use being to set
 94803 up initial display options and alias lists. The following commands shall be invalid in the start-up
 94804 file: **!**, **edit**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **shell**, **visual**, **Copy**, **followup**, and **Followup**.
 94805 Any errors in the start-up file shall either cause *mailx* to terminate with a diagnostic message and
 94806 a non-zero status or to continue after writing a diagnostic message, ignoring the remainder of
 94807 the lines in the start-up file.

94808 A blank line in a start-up file shall be ignored.

94809 Internal Variables in mailx

94810 The following variables are internal *mailx* variables. Each internal variable can be set via the
 94811 *mailx set* command at any time. The **unset** and **set no name** commands can be used to erase
 94812 variables.

94813 In the following list, variables shown as:

94814 *variable*

94815 represent Boolean values. Variables shown as:

94816 *variable=value*

mailx

Utilities

94817		shall be assigned string or numeric values. For string values, the rules in Commands in mailx
94818		(on page 2892) concerning filenames and quoting shall also apply.
94819		The defaults specified here may be changed by the unspecified system start-up file unless the
94820		user specifies the -n option.
94821	allnet	All network names whose login name components match shall be treated as
94822		identical. This shall cause the <i>msglist</i> message specifications to behave similarly.
94823		The default shall be noallnet . See also the alternates command and the metoo
94824		variable.
94825	append	Append messages to the end of the mbox file upon termination instead of placing
94826		them at the beginning. The default shall be noappend . This variable shall not
94827		affect the save command when saving to mbox .
94828	ask, asksub	Prompt for a subject line on outgoing mail if one is not specified on the command
94829		line with the -s option. The ask and asksub forms are synonyms; the system shall
94830		refer to asksub and noasksub in its messages, but shall accept ask and noask as
94831		user input to mean asksub and noasksub . It shall not be possible to set both ask
94832		and noasksub , or noask and asksub . The default shall be asksub , but no
94833		prompting shall be done if standard input is not a terminal.
94834	askbcc	Prompt for the blind copy list. The default shall be noaskbcc .
94835	askcc	Prompt for the copy list. The default shall be noaskcc .
94836	autoprint	Enable automatic writing of messages after delete and undelete commands. The
94837		default shall be noautoprint .
94838	bang	Enable the special-case treatment of <exclamation-mark> characters ('!') in
94839		escape command lines; see the escape command and Command Escapes in mailx
94840		(on page 2901). The default shall be nobang , disabling the expansion of '!' in the
94841		<i>command</i> argument to the ~! command and the ~<!command escape.
94842	cmd=command	
94843		Set the default command to be invoked by the pipe command. The default shall be
94844		nocmd .
94845	crt=number	Pipe messages having more than <i>number</i> lines through the command specified by
94846		the value of the <i>PAGER</i> variable. The default shall be nocrt . If it is set to null, the
94847		value used is implementation-defined.
94848	XS! debug	Enable verbose diagnostics for debugging. Messages are not delivered. The
94849		default shall be nodebug .
94850	dot	When dot is set, a <period> on a line by itself during message input from a
94851		terminal shall also signify end-of-file (in addition to normal end-of-file). The
94852		default shall be nodot . If ignoreeof is set (see below), a setting of nodot shall be
94853		ignored and the <period> is the only method to terminate input mode.
94854	escape=c	Set the command escape character to be the character 'c'. By default, the
94855		command escape character shall be <tilde>. If escape is unset, <tilde> shall be
94856		used; if it is set to null, command escaping shall be disabled.
94857	flipr	Reverse the meanings of the R and r commands. The default shall be noflipr .
94858	folder=directory	
94859		The default directory for saving mail files. User-specified filenames beginning with
94860		a <plus-sign> ('+') shall be expanded by preceding the filename with this

Utilities

mailx

94861		directory name to obtain the real pathname. If <i>directory</i> does not start with a
94862		<slash> (' / '), the contents of <i>HOME</i> shall be prefixed to it. The default shall be
94863		nofolder . If folder is unset or set to null, user-specified filenames beginning with
94864		'+' shall refer to files in the current directory that begin with the literal '+'
94865		character. See also outfolder below. The folder value need not affect the processing
94866		of the files named in <i>MBOX</i> and <i>DEAD</i> .
94867	header	Enable writing of the header summary when entering <i>mailx</i> in Receive Mode. The
94868		default shall be header .
94869	hold	Preserve all messages that are read in the system mailbox instead of putting them
94870		in the mbox save file. The default shall be nohold .
94871	ignore	Ignore interrupts while entering messages. The default shall be noignore .
94872	ignoreeof	Ignore normal end-of-file during message input. Input can be terminated only by
94873		entering a <period> (' . ') on a line by itself or by the ~. command escape. The
94874		default shall be noignoreeof . See also dot above.
94875	indentprefix=string	
94876		A string that shall be added as a prefix to each line that is inserted into the message
94877		by the ~m command escape. This variable shall default to one <tab>.
94878	keep	When a system mailbox, secondary mailbox, or mbox is empty, truncate it to zero
94879		length instead of removing it. The default shall be nokeep .
94880	keepsave	Keep the messages that have been saved from the system mailbox into other files
94881		in the file designated by the variable <i>MBOX</i> , instead of deleting them. The default
94882		shall be nokeepsave .
94883	metoo	Suppress the deletion of the login name of the user from the recipient list when
94884		replying to a message or sending to a group. The default shall be nometoo .
94885	XSI onehop	When responding to a message that was originally sent to several recipients, the
94886		other recipient addresses are normally forced to be relative to the originating
94887		author's machine for the response. This flag disables alteration of the recipients'
94888		addresses, improving efficiency in a network where all machines can send directly
94889		to all other machines (that is, one hop away). The default shall be noonehop .
94890	outfolder	Cause the files used to record outgoing messages to be located in the directory
94891		specified by the folder variable unless the pathname is absolute. The default shall
94892		be nooutfolder . See the record variable.
94893	page	Insert a <form-feed> after each message sent through the pipe created by the pipe
94894		command. The default shall be nopage .
94895	prompt=string	
94896		Set the command-mode prompt to <i>string</i> . If <i>string</i> is null or if noprompt is set, no
94897		prompting shall occur. The default shall be to prompt with the string "? ".
94898	quiet	Refrain from writing the opening message and version when entering <i>mailx</i> . The
94899		default shall be noquiet .
94900	record=file	Record all outgoing mail in the file with the pathname <i>file</i> . The default shall be
94901		norecord . See also outfolder above.
94902	save	Enable saving of messages in the dead-letter file on interrupt or delivery error. See
94903		the variable <i>DEAD</i> for the location of the dead-letter file. The default shall be save .

mailx*Utilities*

94904	screen=number	
94905		Set the number of lines in a screenful of headers for the headers and z commands.
94906		If screen is not specified, a value based on the terminal type identified by the
94907		<i>TERM</i> environment variable, the window size, the baud rate, or some combination
94908		of these shall be used.
94909	sendwait	Wait for the background mailer to finish before returning. The default shall be
94910		nosendwait .
94911	showto	When the sender of the message was the user who is invoking <i>mailx</i> , write the
94912		information from the To: line instead of the From: line in the header summary. The
94913		default shall be noshowto .
94914	sign=string	Set the variable inserted into the text of a message when the ~a command escape is
94915		given. The default shall be noSign . The character sequences ' \t ' and ' \n ' shall
94916		be recognized in the variable as <tab> and <newline> characters, respectively. (See
94917		also ~i in Command Escapes in mailx (on page 2901).)
94918	Sign=string	Set the variable inserted into the text of a message when the ~A command escape is
94919		given. The default shall be noSign . The character sequences ' \t ' and ' \n ' shall
94920		be recognized in the variable as <tab> and <newline> characters, respectively.
94921	toplines=number	
94922		Set the number of lines of the message to write with the top command. The default
94923		shall be 5.

Commands in mailx

The following *mailx* commands shall be provided. In the following list, header refers to lines from the message header, as shown in the OUTPUT FILES section. Header-line refers to lines within the header that begin with one or more non-white-space characters, immediately followed by a <colon> and white space and continuing until the next line beginning with a non-white-space character or an empty line. Header-field refers to the portion of a header line prior to the first <colon> in that line.

For each of the commands listed below, the command can be entered as the abbreviation (those characters in the Synopsis command word preceding the '**[**'), the full command (all characters shown for the command word, omitting the '**[**' and '**]**'), or any truncation of the full command down to the abbreviation. For example, the **exit** command (shown as **ex[it]** in the Synopsis) can be entered as **ex**, **exi**, or **exit**.

The arguments to commands can be quoted, using the following methods:

- An argument can be enclosed between paired double-quotes ("**"**") or single-quotes ('**'**'); any white space, shell word expansion, or <backslash> characters within the quotes shall be treated literally as part of the argument. A double-quote shall be treated literally within single-quotes and *vice versa*. These special properties of the <quotation-mark> characters shall occur only when they are paired at the beginning and end of the argument.
- A <backslash> outside of the enclosing quotes shall be discarded and the following character treated literally as part of the argument.
- An unquoted <backslash> at the end of a command line shall be discarded and the next line shall continue the command.

94946 Filenames, where expected, shall be subjected to the following transformations, in sequence:

- 94947 • If the filename begins with an unquoted <plus-sign>, and the **folder** variable is defined
- 94948 (see the **folder** variable), the <plus-sign> shall be replaced by the value of the **folder**
- 94949 variable followed by a <slash>. If the **folder** variable is unset or is set to null, the filename
- 94950 shall be unchanged.
- 94951 • Shell word expansions shall be applied to the filename (see [Section 2.6](#), on page 2305). If
- 94952 more than a single pathname results from this expansion and the command is expecting
- 94953 one file, the effects are unspecified.

94954 Declare Aliases

94955 *Synopsis:* a[lias] [alias [address...]]

94956 g[roup] [alias [address...]]

94957 Add the given addresses to the alias specified by *alias*. The names shall be substituted when

94958 *alias* is used as a recipient address specified by the user in an outgoing message (that is, other

94959 recipients addressed indirectly through the **reply** command shall not be substituted in this

94960 manner). Mail address alias substitution shall apply only when the alias string is used as a full

94961 address; for example, when **hlj** is an alias, *hlj@posix.com* does not trigger the alias substitution. If

94962 no arguments are given, write a listing of the current aliases to standard output. If only an *alias*

94963 argument is given, write a listing of the specified alias to standard output. These listings need

94964 not reflect the same order of addresses that were entered.

94965 Declare Alternatives

94966 *Synopsis:* alt[ernates] name...

94967 (See also the **metoo** variable.) Declare a list of alternative names for the user's login. When

94968 responding to a message, these names shall be removed from the list of recipients for the

94969 response. The comparison of names shall be in a case-insensitive manner. With no arguments,

94970 **alternates** shall write the current list of alternative names.

94971 Change Current Directory

94972 *Synopsis:* cd [directory]

94973 ch[dir] [directory]

94974 Change directory. If *directory* is not specified, the contents of *HOME* shall be used.

94975 Copy Messages

94976 *Synopsis:* c[opy] [file]

94977 c[opy] [msglist] file

94978 C[opy] [msglist]

94979 Copy messages to the file named by the pathname *file* without marking the messages as saved.

94980 Otherwise, it shall be equivalent to the **save** command.

94981 In the capitalized form, save the specified messages in a file whose name is derived from the

94982 author of the message to be saved, without marking the messages as saved. Otherwise, it shall

94983 be equivalent to the **Save** command.

Delete Messages

Synopsis: d[delete] [msglist]

Mark messages for deletion from the mailbox. The deletions shall not occur until *mailx* quits (see the **quit** command) or changes mailboxes (see the **folder** command). If **autoprint** is set and there are messages remaining after the **delete** command, the current message shall be written as described for the **print** command (see the **print** command); otherwise, the *mailx* prompt shall be written.

Discard Header Fields

Synopsis: di[scard] [header-field...]
 ig[nore] [header-field...]

Suppress the specified header fields when writing messages. Specified *header-fields* shall be added to the list of suppressed header fields. Examples of header fields to ignore are **status** and **cc**. The fields shall be included when the message is saved. The **Print** and **Type** commands shall override this command. The comparison of header fields shall be in a case-insensitive manner. If no arguments are specified, write a list of the currently suppressed header fields to standard output; the listing need not reflect the same order of header fields that were entered.

If both **retain** and **discard** commands are given, **discard** commands shall be ignored.

Delete Messages and Display

Synopsis: dp [msglist]
 dt [msglist]

Delete the specified messages as described for the **delete** command, except that the **autoprint** variable shall have no effect, and the current message shall be written only if it was set to a message after the last message deleted by the command. Otherwise, an informational message to the effect that there are no further messages in the mailbox shall be written, followed by the *mailx* prompt.

Echo a String

Synopsis: ec[ho] *string* ...

Echo the given strings, equivalent to the shell *echo* utility.

Edit Messages

Synopsis: e[dit] [msglist]

Edit the given messages. The messages shall be placed in a temporary file and the utility named by the *EDITOR* variable is invoked to edit each file in sequence. The default *EDITOR* is unspecified.

The **edit** command does not modify the contents of those messages in the mailbox.

Exit

Synopsis: ex[it]

 x[it]

Exit from *mailx* without changing the mailbox. No messages shall be saved in the **mbox** (see also **quit**).

Change Folder

Synopsis: fi[le] [*file*]

 fold[er] [*file*]

Quit (see the **quit** command) from the current file of messages and read in the file named by the pathname *file*. If no argument is given, the name and status of the current mailbox shall be written.

Several unquoted special characters shall be recognized when used as *file* names, with the following substitutions:

% The system mailbox for the invoking user.

%*user* The system mailbox for *user*.

The previous file.

& The current **mbox**.

+*file* The named file in the **folder** directory. (See the **folder** variable.)

The default file shall be the current mailbox.

Display List of Folders

Synopsis: folders

Write the names of the files in the directory set by the **folder** variable. The command specified by the *LISTER* environment variable shall be used (see the ENVIRONMENT VARIABLES section).

Follow Up Specified Messages

Synopsis: fo[llowup] [*message*]

 F[ollowup] [*msglist*]

In the lowercase form, respond to a message, recording the response in a file whose name is derived from the author of the message. See also the **save** and **copy** commands and **outfolder**.

In the capitalized form, respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line shall be taken from the first message and the response shall be recorded in a file whose name is derived from the author of the first message. See also the **Save** and **Copy** commands and **outfolder**.

Both forms shall override the **record** variable, if set.

Display Header Summary for Specified Messages

Synopsis: `f[rom] [msglist]`

Write the header summary for the specified messages.

Display Header Summary

Synopsis: `h[eaders] [message]`

Write the page of headers that includes the message specified. If the *message* argument is not specified, the current message shall not change. However, if the *message* argument is specified, the current message shall become the message that appears at the top of the page of headers that includes the message specified. The **screen** variable sets the number of headers per page. See also the **z** command.

Help

Synopsis: `hel[p]`
 `?`

Write a summary of commands.

Hold Messages

Synopsis: `ho[ld] [msglist]`
 `pre[serve] [msglist]`

Mark the messages in *msglist* to be retained in the mailbox when *mailx* terminates. This shall override any commands that might previously have marked the messages to be deleted. During the current invocation of *mailx*, only the **delete**, **dp**, or **dt** commands shall remove the *preserve* marking of a message.

Execute Commands Conditionally

Synopsis: `i[f] s|r`
 `mail-commands`
 `el[se]`
 `mail-commands`
 `en[dif]`

Execute commands conditionally, where **if s** executes the following *mail-commands*, up to an **else** or **endif**, if the program is in Send Mode, and **if r** shall cause the *mail-commands* to be executed only in Receive Mode.

List Available Commands

Synopsis: `l[ist]`

Write a list of all commands available. No explanation shall be given.

Mail a Message

Synopsis: m[ail] address...

Mail a message to the specified addresses or aliases.

Direct Messages to mbox

Synopsis: mb[ox] [msglist]

Arrange for the given messages to end up in the **mbox** save file when *mailx* terminates normally. See **MBOX**. See also the **exit** and **quit** commands.

Process Next Specified Message

Synopsis: n[ext] [message]

If the current message has not been written (for example, by the **print** command) since *mailx* started or since any other message was the current message, behave as if the **print** command was entered. Otherwise, if there is an undeleted message after the current message, make it the current message and behave as if the **print** command was entered. Otherwise, an informational message to the effect that there are no further messages in the mailbox shall be written, followed by the *mailx* prompt. Should the current message location be the result of an immediately preceding **hold**, **mbox**, **preserve**, or **touch** command, **next** will act as if the current message has already been written.

Pipe Message

Synopsis: pi[pe] [[msglist] command]
 | [[msglist] command]

Pipe the messages through the given *command* by invoking the command interpreter specified by *SHELL* with two arguments: **-c** and *command*. (See also *sh -c*.) The application shall ensure that the command is given as a single argument. Quoting, described previously, can be used to accomplish this. If no arguments are given, the current message shall be piped through the command specified by the value of the **cmd** variable. If the **page** variable is set, a <form-feed> shall be inserted after each message.

Display Message with Headers

Synopsis: P[rint] [msglist]
 T[ype] [msglist]

Write the specified messages, including all header lines, to standard output. Override suppression of lines by the **discard**, **ignore**, and **retain** commands. If **crt** is set, the messages longer than the number of lines specified by the **crt** variable shall be paged through the command specified by the *PAGER* environment variable.

Display Message

Synopsis: p[rint] [msglist]
 t[ype] [msglist]

Write the specified messages to standard output. If **crt** is set, the messages longer than the number of lines specified by the **crt** variable shall be paged through the command specified by the *PAGER* environment variable.

mailx*Utilities***Quit**

Synopsis: q[uit]
 end-of-file

Terminate *mailx*, storing messages that were read in **mbox** (if the current mailbox is the system mailbox and unless **hold** is set), deleting messages that have been explicitly saved (unless **keepsave** is set), discarding messages that have been deleted, and saving all remaining messages in the mailbox.

Reply to a Message List

Synopsis: R[e]ply [msglist]
 R[espond] [msglist]

Mail a reply message to the sender of each message in the *msglist*. The subject line shall be formed by concatenating **Re:**<space> (unless it already begins with that string) and the subject from the first message. If **record** is set to a filename, the response shall be saved at the end of that file.

See also the **flpr** variable.

Reply to a Message

Synopsis: r[e]ply [message]
 r[espond] [message]

Mail a reply message to all recipients included in the header of the message. The subject line shall be formed by concatenating **Re:**<space> (unless it already begins with that string) and the subject from the message. If **record** is set to a filename, the response shall be saved at the end of that file.

See also the **flpr** variable.

Retain Header Fields

Synopsis: ret[ain] [header-field...]

Retain the specified header fields when writing messages. This command shall override all **discard** and **ignore** commands. The comparison of header fields shall be in a case-insensitive manner. If no arguments are specified, write a list of the currently retained header fields to standard output; the listing need not reflect the same order of header fields that were entered.

Save Messages

Synopsis: s[ave] [file]
 s[ave] [msglist] file
 S[ave] [msglist]

Save the specified messages in the file named by the pathname *file*, or the **mbox** if the *file* argument is omitted. The file shall be created if it does not exist; otherwise, the messages shall be appended to the file. The message shall be put in the state *saved*, and shall behave as specified in the description of the *saved* state when the current mailbox is exited by the **quit** or **file** command.

In the capitalized form, save the specified messages in a file whose name is derived from the author of the first message. The name of the file shall be taken to be the author's name with all network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and

95165 **outfolder** variable.

95166 Set Variables

95167 *Synopsis:* **se**[t] [*name*[=*string*]] ...] [*name*=*number* ...] [*noname* ...]

95168 Define one or more variables called *name*. The variable can be given a null, string, or numeric
95169 value. Quoting and <backslash>-escapes can occur anywhere in *string*, as described previously,
95170 as if the *string* portion of the argument were the entire argument. The forms *name* and *name*=
95171 shall be equivalent to *name*="" for variables that take string values. The **set** command without
95172 arguments shall write a list of all defined variables and their values. The **no name** form shall be
95173 equivalent to **unset name**.

95174 Invoke a Shell

95175 *Synopsis:* **sh**[ell]

95176 Invoke an interactive command interpreter (see also *SHELL*).

95177 Display Message Size

95178 *Synopsis:* **si**[ze] [*msglist*]

95179 Write the size in bytes of each of the specified messages.

95180 Read mailx Commands From a File

95181 *Synopsis:* **so**[urce] *file*

95182 Read and execute commands from the file named by the pathname *file* and return to command
95183 mode.

95184 Display Beginning of Messages

95185 *Synopsis:* **to**[p] [*msglist*]

95186 Write the top few lines of each of the specified messages. If the **toplines** variable is set, it is taken
95187 as the number of lines to write. The default shall be 5.

95188 Touch Messages

95189 *Synopsis:* **to**[ch] [*msglist*]

95190 Touch the specified messages. If any message in *msglist* is not specifically deleted nor saved in a
95191 file, it shall be placed in the **mbox** upon normal termination. See **exit** and **quit**.

95192 Delete Aliases

95193 *Synopsis:* **una**[lias] [*alias*] ...

95194 Delete the specified alias names. If a specified alias does not exist, the results are unspecified.

95195 **Undelete Messages**95196 *Synopsis:* u[ndelete] [msglist]

95197 Change the state of the specified messages from deleted to read. If **autoprint** is set, the last
 95198 message of those restored shall be written. If *msglist* is not specified, the message shall be
 95199 selected as follows:

- 95200 • If there are any deleted messages that follow the current message, the first of these shall be
 95201 chosen.
- 95202 • Otherwise, the last deleted message that also precedes the current message shall be chosen.

95203 **Unset Variables**95204 *Synopsis:* uns[et] name...

95205 Cause the specified variables to be erased.

95206 **Edit Message with Full-Screen Editor**95207 *Synopsis:* v[isual] [msglist]

95208 Edit the given messages with a screen editor. Each message shall be placed in a temporary file,
 95209 and the utility named by the *VISUAL* variable shall be invoked to edit each file in sequence. The
 95210 default editor shall be *vi*.

95211 The **visual** command does not modify the contents of those messages in the mailbox.95212 **Write Messages to a File**95213 *Synopsis:* w[rite] [msglist] file

95214 Write the given messages to the file specified by the pathname *file*, minus the message header.
 95215 Otherwise, it shall be equivalent to the **save** command.

95216 **Scroll Header Display**95217 *Synopsis:* z[+|-]

95218 Scroll the header display forward (if '+' is specified or if no option is specified) or backward (if
 95219 '-' is specified) one screenful. The number of headers written shall be set by the **screen**
 95220 variable.

95221 **Invoke Shell Command**95222 *Synopsis:* !command

95223 Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and *command*. (See
 95224 also *sh -c*.) If the **bang** variable is set, each unescaped occurrence of '!' in *command* shall be
 95225 replaced with the command executed by the previous ! command or ~! command escape.

Null Command

Synopsis: # *comment*

This null command (*comment*) shall be ignored by *mailx*.

Display Current Message Number

Synopsis: =

Write the current message number.

Command Escapes in mailx

The following commands can be entered only from input mode, by beginning a line with the escape character (by default, <tilde> ('~')). See the **escape** variable description for changing this special character. The format for the commands shall be:

<escape-character><command-char><separator> [<arguments>]

where the <separator> can be zero or more <blank> characters.

In the following descriptions, the application shall ensure that the argument *command* (but not *mailx-command*) is a shell command string. Any string acceptable to the command interpreter specified by the *SHELL* variable when it is invoked as *SHELL -c command_string* shall be valid. The command can be presented as multiple arguments (that is, quoting is not required).

Command escapes that are listed with *msglist* or *mailx-command* arguments are invalid in Send Mode and produce unspecified results.

~! *command* Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and *command*; and then return to input mode. If the **bang** variable is set, each unescaped occurrence of '!' in *command* shall be replaced with the command executed by the previous **!** command or **~!** command escape.

~. Simulate end-of-file (terminate message input).

~: *mailx-command*, **~_ *mailx-command***
Perform the command-level request.

~? Write a summary of command escapes.

~A This shall be equivalent to **~i Sign**.

~a This shall be equivalent to **~i sign**.

~b *name...* Add the *names* to the blind carbon copy (**Bcc**) list.

~c *name...* Add the *names* to the carbon copy (**Cc**) list.

~d Read in the dead-letter file. See *DEAD* for a description of this file.

~e Invoke the editor, as specified by the *EDITOR* environment variable, on the partial message.

~f [*msglist*] Forward the specified messages. The specified messages shall be inserted into the current message without alteration. This command escape also shall insert message headers into the message with field selection affected by the **discard**, **ignore**, and **retain** commands.

mailx

Utilities

95263	~F [msglist]	This shall be the equivalent of the ~f command escape, except that all headers shall
95264		be included in the message, regardless of previous discard , ignore , and retain
95265		commands.
95266	~h	If standard input is a terminal, prompt for a Subject line and the To , Cc , and Bcc
95267		lists. Other implementation-defined headers may also be presented for editing. If
95268		the field is written with an initial value, it can be edited as if it had just been typed.
95269	~i string	Insert the value of the named variable, followed by a <newline>, into the text of
95270		the message. If the string is unset or null, the message shall not be changed.
95271	~m [msglist]	Insert the specified messages into the message, prefixing non-empty lines with the
95272		string in the indentprefix variable. This command escape also shall insert message
95273		headers into the message, with field selection affected by the discard , ignore , and
95274		retain commands.
95275	~M [msglist]	This shall be the equivalent of the ~m command escape, except that all headers
95276		shall be included in the message, regardless of previous discard , ignore , and retain
95277		commands.
95278	~p	Write the message being entered. If the message is longer than crt lines (see
95279		Internal Variables in mailx , on page 2889), the output shall be paginated as
95280		described for the PAGER variable.
95281	~q	Quit (see the quit command) from input mode by simulating an interrupt. If the
95282		body of the message is not empty, the partial message shall be saved in the dead-
95283		letter file. See DEAD for a description of this file.
95284	~r file, ~< file, ~r !command, ~< !command	
95285		Read in the file specified by the pathname <i>file</i> . If the argument begins with an
95286		<exclamation-mark> ('!'), the rest of the string shall be taken as an arbitrary
95287		system command; the command interpreter specified by SHELL shall be invoked
95288		with two arguments: -c and <i>command</i> . The standard output of <i>command</i> shall be
95289		inserted into the message.
95290	~s string	Set the subject line to <i>string</i> .
95291	~t name...	Add the given <i>names</i> to the To list.
95292	~v	Invoke the full-screen editor, as specified by the VISUAL environment variable, on
95293		the partial message.
95294	~w file	Write the partial message, without the header, onto the file named by the
95295		pathname <i>file</i> . The file shall be created or the message shall be appended to it if
95296		the file exists.
95297	~x	Exit as with ~q, except the message shall not be saved in the dead-letter file.
95298	~ command	Pipe the body of the message through the given <i>command</i> by invoking the
95299		command interpreter specified by SHELL with two arguments: -c and <i>command</i> . If
95300		the <i>command</i> returns a successful exit status, the standard output of the command
95301		shall replace the message. Otherwise, the message shall remain unchanged. If the
95302		<i>command</i> fails, an error message giving the exit status shall be written.

95303 **EXIT STATUS**95304 UP When the **-e** option is specified, the following exit values are returned:

95305 0 Mail was found.

95306 >0 Mail was not found or an error occurred.

95307 Otherwise, the following exit values are returned:

95308 0 Successful completion; note that this status implies that all messages were *sent*, but it gives
95309 no assurances that any of them were actually *delivered*.

95310 >0 An error occurred.

95311 **CONSEQUENCES OF ERRORS**95312 UP When in **input mode (Receive Mode)** or **Send Mode**:95313 • If an error is encountered processing an input line beginning with a <tilde> ('~')
95314 UP character, (see **Command Escapes in mailx**, on page 2901), a diagnostic message shall be
95315 written to standard error, and the message being composed may be modified, but this
95316 condition shall not prevent the message from being sent.

95317 • Other errors shall prevent the sending of the message.

95318 UP When in **command mode**:

95319 • Default.

95320 **APPLICATION USAGE**95321 Delivery of messages to remote systems requires the existence of communication paths to such
95322 systems. These need not exist.95323 Input lines are limited to {LINE_MAX} bytes, but mailers between systems may impose more
95324 severe line-length restrictions. This volume of POSIX.1-2008 does not place any restrictions on
95325 the length of messages handled by *mailx*, and for delivery of local messages the only limitations
95326 should be the normal problems of available disk space for the target mail file. When sending
95327 messages to external machines, applications are advised to limit messages to less than 100 000
95328 bytes because some mail gateways impose message-length restrictions.95329 The format of the system mailbox is intentionally unspecified. Not all systems implement
95330 system mailboxes as flat files, particularly with the advent of multimedia mail messages. Some
95331 system mailboxes may be multiple files, others records in a database. The internal format of the
95332 messages themselves is specified with the historical format from Version 7, but only after the
95333 messages have been saved in some file other than the system mailbox. This was done so that
95334 many historical applications expecting text-file mailboxes are not broken.95335 Some new formats for messages can be expected in the future, probably including binary data,
95336 bit maps, and various multimedia objects. As described here, *mailx* is not prohibited from
95337 handling such messages, but it must store them as text files in secondary mailboxes (unless some
95338 extension, such as a variable or command line option, is used to change the stored format). Its
95339 method of doing so is implementation-defined and might include translating the data into text
95340 file-compatible or readable form or omitting certain portions of the message from the stored
95341 output.95342 The **discard** and **ignore** commands are not inverses of the **retain** command. The **retain**
95343 command discards all header-fields except those explicitly retained. The **discard** command
95344 keeps all header-fields except those explicitly discarded. If headers exist on the retained header

mailx*Utilities*

95345 list, **discard** and **ignore** commands are ignored.

95346 **EXAMPLES**

95347 None.

95348 **RATIONALE**

95349 The standard developers felt strongly that a method for applications to send messages to specific
 95350 users was necessary. The obvious example is a batch utility, running non-interactively, that
 95351 wishes to communicate errors or results to a user. However, the actual format, delivery
 95352 mechanism, and method of reading the message are clearly beyond the scope of this volume of
 95353 POSIX.1-2008.

95354 The intent of this command is to provide a simple, portable interface for sending messages non-
 95355 interactively. It merely defines a “front-end” to the historical mail system. It is suggested that
 95356 implementations explicitly denote the sender and recipient in the body of the delivered message.
 95357 Further specification of formats for either the message envelope or the message itself were
 95358 deliberately not made, as the industry is in the midst of changing from the current standards to a
 95359 more internationalized standard and it is probably incorrect, at this time, to require either one.

95360 Implementations are encouraged to conform to the various delivery mechanisms described in
 95361 the CCITT X.400 standards or to the equivalent Internet standards, described in Internet Request
 95362 for Comment (RFC) documents RFC 819, RFC 822, RFC 920, RFC 921, and RFC 1123.

95363 Many historical systems modified each body line that started with **From** by prefixing the ‘F’
 95364 with ‘>’. It is unnecessary, but allowed, to do that when the string does not follow a blank line
 95365 because it cannot be confused with the next header.

95366 The **edit** and **visual** commands merely edit the specified messages in a temporary file. They do
 95367 not modify the contents of those messages in the mailbox; such a capability could be added as an
 95368 extension, such as by using different command names.

95369 The restriction on a subject line being {LINE_MAX}–10 bytes is based on the historical format
 95370 that consumes 10 bytes for **Subject** and the trailing <newline>. Many historical mailers that a
 95371 message may encounter on other systems are not able to handle lines that long, however.

95372 Like the utilities *logger* and *lp*, *mailx* admittedly is difficult to test. This was not deemed sufficient
 95373 justification to exclude this utility from this volume of POSIX.1-2008. It is also arguable that it is,
 95374 in fact, testable, but that the tests themselves are not portable.

95375 When *mailx* is being used by an application that wishes to receive the results as if none of the
 95376 User Portability Utilities option features were supported, the *DEAD* environment variable must
 95377 be set to **/dev/null**. Otherwise, it may be subject to the file creations described in *mailx*
 95378 ASYNCHRONOUS EVENTS. Similarly, if the *MAILRC* environment variable is not set to
 95379 **/dev/null**, historical versions of *mailx* and *Mail* read initialization commands from a file before
 95380 processing begins. Since the initialization that a user specifies could alter the contents of
 95381 messages an application is trying to send, such applications must set *MAILRC* to **/dev/null**.

95382 The description of *LC_TIME* uses “may affect” because many historical implementations do not
 95383 or cannot manipulate the date and time strings in the incoming mail headers. Some headers
 95384 found in incoming mail do not have enough information to determine the timezone in which the
 95385 mail originated, and, therefore, *mailx* cannot convert the date and time strings into the internal
 95386 form that then is parsed by routines like *strftime()* that can take *LC_TIME* settings into account.
 95387 Changing all these times to a user-specified format is allowed, but not required.

95388 The paginator selected when *PAGER* is null or unset is partially unspecified to allow the System
 95389 V historical practice of using *pg* as the default. Bypassing the pagination function, such as by
 95390 declaring that *cat* is the paginator, would not meet with the intended meaning of this

description. However, any “portable user” would have to set *PAGER* explicitly to get his or her preferred paginator on all systems. The paginator choice was made partially unspecified, unlike the *VISUAL* editor choice (mandated to be *vi*) because most historical pagers follow a common theme of user input, whereas editors differ dramatically.

Options to specify addresses as **cc** (carbon copy) or **bcc** (blind carbon copy) were considered to be format details and were omitted.

A zero exit status implies that all messages were *sent*, but it gives no assurances that any of them were actually *delivered*. The reliability of the delivery mechanism is unspecified and is an appropriate marketing distinction between systems.

In order to conform to the Utility Syntax Guidelines, a solution was required to the optional *file* option-argument to **-f**. By making *file* an operand, the guidelines are satisfied and users remain portable. However, it does force implementations to support usage such as:

```
mailx -fin mymail.box
```

The **no name** method of unsetting variables is not present in all historical systems, but it is in System V and provides a logical set of commands corresponding to the format of the display of options from the *mailx set* command without arguments.

The **ask** and **asksub** variables are the names selected by BSD and System V, respectively, for the same feature. They are synonyms in this volume of POSIX.1-2008.

The *mailx echo* command was not documented in the BSD version and has been omitted here because it is not obviously useful for interactive users.

The default prompt on the System V *mailx* is a <question-mark>, on BSD *Mail* an <ampersand>. Since this volume of POSIX.1-2008 chose the *mailx* name, it kept the System V default, assuming that BSD users would not have difficulty with this minor incompatibility (that they can override).

The meanings of **r** and **R** are reversed between System V *mailx* and SunOS *Mail*. Once again, since this volume of POSIX.1-2008 chose the *mailx* name, it kept the System V default, but allows the SunOS user to achieve the desired results using **flpr**, an internal variable in System V *mailx*, although it has not been documented in the SVID.

The **indentprefix** variable, the **retain** and **unalias** commands, and the **~F** and **~M** command escapes were adopted from 4.3 BSD *Mail*.

The **version** command was not included because no sufficiently general specification of the version information could be devised that would still be useful to a portable user. This command name should be used by suppliers who wish to provide version information about the *mailx* command.

The “implementation-specific (unspecified) system start-up file” historically has been named **/etc/mailx.rc**, but this specific name and location are not required.

The intent of the wording for the **next** command is that if any command has already displayed the current message it should display a following message, but, otherwise, it should display the current message. Consider the command sequence:

```
next 3
delete 3
next
```

where the **autoprint** option was not set. The normative text specifies that the second **next** command should display a message following the third message, because even though the

95435 current message has not been displayed since it was set by the **delete** command, it has been
 95436 displayed since the current message was anything other than message number 3. This does not
 95437 always match historical practice in some implementations, where the command file address
 95438 followed by **next** (or the default command) would skip the message for which the user had
 95439 searched.

95440 FUTURE DIRECTIONS

95441 None.

95442 SEE ALSO

95443 [Chapter 2](#) (on page 2297), *ed*, *ls*, *more*, *vi*

95444 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

95445 CHANGE HISTORY

95446 First released in Issue 2.

95447 Issue 5

95448 The description of the *EDITOR* environment variable is changed to indicate that *ed* is the default
 95449 editor if this variable is not set. In previous issues, this default was not stated explicitly at this
 95450 point but was implied further down in the text.

95451 The FUTURE DIRECTIONS section is added.

95452 Issue 6

95453 The following new requirements on POSIX implementations derive from alignment with the
 95454 Single UNIX Specification:

- 95455 • The **-F** option is added.
- 95456 • The **allnet**, **debug**, and **sendwait** internal variables are added.
- 95457 • The **C**, **ec**, **fo**, **F**, and **S** *mailx* commands are added.

95458 In the DESCRIPTION and ENVIRONMENT VARIABLES sections, text stating “HOME
 95459 directory” is replaced by “directory referred to by the HOME environment variable”.

95460 The *mailx* utility is aligned with the IEEE P1003.2b draft standard, which includes various
 95461 clarifications to resolve IEEE PASC Interpretations submitted for the ISO POSIX-2:1993
 95462 standard. In particular, the changes here address IEEE PASC Interpretations 1003.2 #10, #11,
 95463 #103, #106, #108, #114, #115, #122, and #129.

95464 The normative text is reworded to avoid use of the term “must” for application requirements.

95465 The TZ entry is added to the ENVIRONMENT VARIABLES section.

95466 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/32 is applied, applying a change to the
 95467 EXTENDED DESCRIPTION, raised by IEEE PASC Interpretation 1003.2 #122, which was
 95468 overlooked in the first version of this standard.

95469 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/17 is applied, updating the EXTENDED
 95470 DESCRIPTION (Internal Variables in *mailx*). The system start-up file is changed from
 95471 “implementation-defined” to “unspecified” for consistency with other text in the EXTENDED
 95472 DESCRIPTION.

95473 Issue 7

95474 Austin Group Interpretation 1003.1-2001 #089 is applied, clarifying the effect of the *LC_TIME*
 95475 environment variable.

95476 Austin Group Interpretation 1003.1-2001 #090 is applied, updating the description of the **next**
 95477 command.

- 95478 SD5-XCU-ERN-69 is applied.
- 95479 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 95480 Shading to indicate support for the User Portability Utilities option is added.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

make

Utilities

95481 **NAME**95482 **make** — maintain, update, and regenerate groups of programs (**DEVELOPMENT**)95483 **SYNOPSIS**

```
95484 SD  make [-einpqrst] [-f makefile]... [-k|-S] [macro=value...]
95485      [target_name...]
```

95486 **DESCRIPTION**

95487 The *make* utility shall update files that are derived from other files. A typical case is one where
 95488 object files are derived from the corresponding source files. The *make* utility examines time
 95489 relationships and shall update those derived files (called targets) that have modified times
 95490 earlier than the modified times of the files (called prerequisites) from which they are derived. A
 95491 description file (makefile) contains a description of the relationships between files, and the
 95492 commands that need to be executed to update the targets to reflect changes in their
 95493 prerequisites. Each specification, or rule, shall consist of a target, optional prerequisites, and
 95494 optional commands to be executed when a prerequisite is newer than the target. There are two
 95495 types of rule:

- 95496 1. *Inference rules*, which have one target name with at least one <period> ('.') and no
 95497 <slash> ('/')
- 95498 2. *Target rules*, which can have more than one target name

95499 In addition, *make* shall have a collection of built-in macros and inference rules that infer
 95500 prerequisite relationships to simplify maintenance of programs.

95501 To receive exactly the behavior described in this section, the user shall ensure that a portable
 95502 makefile shall:

- 95503 • Include the special target **.POSIX**
- 95504 • Omit any special target reserved for implementations (a leading period followed by
 95505 uppercase letters) that has not been specified by this section

95506 The behavior of *make* is unspecified if either or both of these conditions are not met.

95507 **OPTIONS**

95508 The *make* utility shall conform to XBD [Section 12.2](#) (on page 215), except for Guideline 9.

95509 The following options shall be supported:

- 95510 **-e** Cause environment variables, including those with null values, to override macro
 95511 assignments within makefiles.
- 95512 **-f *makefile*** Specify a different makefile. The argument *makefile* is a pathname of a description
 95513 file, which is also referred to as the *makefile*. A pathname of '-' shall denote the
 95514 standard input. There can be multiple instances of this option, and they shall be
 95515 processed in the order specified. The effect of specifying the same option-argument
 95516 more than once is unspecified.
- 95517 **-i** Ignore error codes returned by invoked commands. This mode is the same as if the
 95518 special target **.IGNORE** were specified without prerequisites.
- 95519 **-k** Continue to update other targets that do not depend on the current target if a non-
 95520 ignored error occurs while executing the commands to bring a target up-to-date.
- 95521 **-n** Write commands that would be executed on standard output, but do not execute
 95522 them. However, lines with a <plus-sign> ('+') prefix shall be executed. In this
 95523 mode, lines with an at-sign ('@') character prefix shall be written to standard

Utilities

make

- 95524 output.
- 95525 **-p** Write to standard output the complete set of macro definitions and target
95526 descriptions. The output format is unspecified.
- 95527 **-q** Return a zero exit value if the target file is up-to-date; otherwise, return an exit
95528 value of 1. Targets shall not be updated if this option is specified. However, a
95529 makefile command line (associated with the targets) with a <plus-sign> ('+')
95530 prefix shall be executed.
- 95531 **-r** Clear the suffix list and do not use the built-in rules.
- 95532 **-S** Terminate *make* if an error occurs while executing the commands to bring a target
95533 up-to-date. This shall be the default and the opposite of **-k**.
- 95534 **-s** Do not write makefile command lines or touch messages (see **-t**) to standard
95535 output before executing. This mode shall be the same as if the special target
95536 **.SILENT** were specified without prerequisites.
- 95537 **-t** Update the modification time of each target as though a *touch target* had been
95538 executed. Targets that have prerequisites but no commands (see **Target Rules**, on
95539 page 2913), or that are already up-to-date, shall not be touched in this manner.
95540 Write messages to standard output for each target file indicating the name of the
95541 file and that it was touched. Normally, the *makefile* command lines associated with
95542 each target are not executed. However, a command line with a <plus-sign> ('+')
95543 prefix shall be executed.

95544 Any options specified in the *MAKEFLAGS* environment variable shall be evaluated before any
95545 options specified on the *make* utility command line. If the **-k** and **-S** options are both specified
95546 on the *make* utility command line or by the *MAKEFLAGS* environment variable, the last option
95547 specified shall take precedence. If the **-f** or **-p** options appear in the *MAKEFLAGS* environment
95548 variable, the result is undefined.

95549 **OPERANDS**

95550 The following operands shall be supported:

95551 *target_name* Target names, as defined in the EXTENDED DESCRIPTION section. If no target is
95552 specified, while *make* is processing the makefiles, the first target that *make*
95553 encounters that is not a special target or an inference rule shall be used.

95554 *macro=value* Macro definitions, as defined in **Macros** (on page 2914).

95555 If the *target_name* and *macro=value* operands are intermixed on the *make* utility command line,
95556 the results are unspecified.

95557 **STDIN**

95558 The standard input shall be used only if the *makefile* option-argument is '-'. See the INPUT
95559 FILES section.

95560 **INPUT FILES**

95561 The input file, otherwise known as the makefile, is a text file containing rules, macro definitions,
95562 and comments. See the EXTENDED DESCRIPTION section.

95563 **ENVIRONMENT VARIABLES**

95564 The following environment variables shall affect the execution of *make*:

95565 **LANG** Provide a default value for the internationalization variables that are unset or null.
95566 (See XBD **Section 8.2** (on page 174) for the precedence of internationalization
95567 variables used to determine the values of locale categories.)

make*Utilities*

95568		LC_ALL	If set to a non-empty string value, override the values of all the other internationalization variables.
95569			
95570		LC_CTYPE	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).
95571			
95572			
95573		LC_MESSAGES	
95574			Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
95575			
95576		MAKEFLAGS	
95577			This variable shall be interpreted as a character string representing a series of option characters to be used as the default options. The implementation shall accept both of the following formats (but need not accept them when intermixed):
95578			
95579			
95580			• The characters are option letters without the leading <hyphen> characters or <blank> separation used on a <i>make</i> utility command line.
95581			
95582			• The characters are formatted in a manner similar to a portion of the <i>make</i> utility command line: options are preceded by <hyphen> characters and <blank>-separated as described in XBD Section 12.2 (on page 215). The <i>macro=value</i> macro definition operands can also be included. The difference between the contents of MAKEFLAGS and the <i>make</i> utility command line is that the contents of the variable shall not be subjected to the word expansions (see Section 2.6, on page 2305) associated with parsing the command line values.
95583			
95584			
95585			
95586			
95587			
95588			
95589			
95590	XSI	NLSPATH	Determine the location of message catalogs for the processing of LC_MESSAGES .
95591	XSI	PROJECTDIR	
95592			Provide a directory to be used to search for SCCS files not found in the current directory. In all of the following cases, the search for SCCS files is made in the directory SCCS in the identified directory. If the value of PROJECTDIR begins with a <slash>, it shall be considered an absolute pathname; otherwise, the value of PROJECTDIR is treated as a user name and that user's initial working directory shall be examined for a subdirectory src or source . If such a directory is found, it shall be used. Otherwise, the value is used as a relative pathname.
95593			
95594			
95595			
95596			
95597			
95598			
95599			If PROJECTDIR is not set or has a null value, the search for SCCS files shall be made in the directory SCCS in the current directory.
95600			
95601			The setting of PROJECTDIR affects all files listed in the remainder of this utility description for files with a component named SCCS .
95602			
95603			The value of the SHELL environment variable shall not be used as a macro and shall not be modified by defining the SHELL macro in a makefile or on the command line. All other environment variables, including those with null values, shall be used as macros, as defined in Macros (on page 2914).
95604			
95605			
95606			
95607		ASYNCHRONOUS EVENTS	
95608			If not already ignored, <i>make</i> shall trap SIGHUP , SIGTERM , SIGINT , and SIGQUIT and remove the current target unless the target is a directory or the target is a prerequisite of the special target .PRECIOUS or unless one of the -n , -p , or -q options was specified. Any targets removed in this manner shall be reported in diagnostic messages of unspecified format, written to standard error. After this cleanup process, if any, <i>make</i> shall take the standard action for all other signals.
95609			
95610			
95611			
95612			
95613			

STDOUT

The *make* utility shall write all commands to be executed to standard output unless the `-s` option was specified, the command is prefixed with an at-sign, or the special target **.SILENT** has either the current target as a prerequisite or has no prerequisites. If *make* is invoked without any work needing to be done, it shall write a message to standard output indicating that no action was taken. If the `-t` option is present and a file is touched, *make* shall write to standard output a message of unspecified format indicating that the file was touched, including the filename of the file.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

Files can be created when the `-t` option is present. Additional files can also be created by the utilities invoked by *make*.

EXTENDED DESCRIPTION

The *make* utility attempts to perform the actions required to ensure that the specified targets are up-to-date. A target is considered out-of-date if it is older than any of its prerequisites or if it does not exist. The *make* utility shall treat all prerequisites as targets themselves and recursively ensure that they are up-to-date, processing them in the order in which they appear in the rule. The *make* utility shall use the modification times of files to determine whether the corresponding targets are out-of-date.

After *make* has ensured that all of the prerequisites of a target are up-to-date and if the target is out-of-date, the commands associated with the target entry shall be executed. If there are no commands listed for the target, the target shall be treated as up-to-date.

Makefile Syntax

A makefile can contain rules, macro definitions (see **Macros**, on page 2914), include lines, and comments. There are two kinds of rules: *inference rules* and *target rules*. The *make* utility shall contain a set of built-in inference rules. If the `-r` option is present, the built-in rules shall not be used and the suffix list shall be cleared. Additional rules of both types can be specified in a makefile. If a rule is defined more than once, the value of the rule shall be that of the last one specified. Macros can also be defined more than once, and the value of the macro is specified in **Macros** (on page 2914). Comments start with a <number-sign> ('#') and continue until an unescaped <newline> is reached.

By default, the following files shall be tried in sequence: **.makefile** and **.Makefile**. If neither **.makefile** or **.Makefile** are found, other implementation-defined files may also be tried. On XSI-conformant systems, the additional files **.s.makefile**, **SCCS/s.makefile**, **.s.Makefile**, and **SCCS/s.Makefile** shall also be tried.

The `-f` option shall direct *make* to ignore any of these default files and use the specified argument as a makefile instead. If the `'-'` argument is specified, standard input shall be used.

The term *makefile* is used to refer to any rules provided by the user, whether in **.makefile** or its variants, or specified by the `-f` option.

The rules in makefiles shall consist of the following types of lines: target rules, including special targets (see **Target Rules**, on page 2913), inference rules (see **Inference Rules**, on page 2916), macro definitions (see **Macros**, on page 2914), empty lines, and comments.

Target and Inference Rules may contain *command lines*. Command lines can have a prefix that shall be removed before execution (see **Makefile Execution**, on page 2912).

When an escaped <newline> (one preceded by a <backslash>) is found anywhere in the makefile except in a command line, an include line, or a line immediately preceding an include line, it shall be replaced, along with any leading white space on the following line, with a single <space>. When an escaped <newline> is found in a command line in a makefile, the command line shall contain the <backslash>, the <newline>, and the next line, except that the first character of the next line shall not be included if it is a <tab>. When an escaped <newline> is found in an include line or in a line immediately preceding an include line, the behavior is unspecified.

Include Lines

If the word **include** appears at the beginning of a line and is followed by one or more <blank> characters, the string formed by the remainder of the line shall be processed as follows to produce a pathname:

- The trailing <newline> and any comment shall be discarded. If the resulting string contains any double-quote characters (' " ') the behavior is unspecified.
- The resulting string shall be processed for macro expansion (see **Macros** (on page 2914)).
- Any <blank> characters that appear after the first non-<blank> shall be used as separators to divide the macro-expanded string into fields. It is unspecified whether any other white-space characters are also used as separators. It is unspecified whether pathname expansion (see **Section 2.13**, on page 2332) is also performed.
- If the processing of separators and optional pathname expansion results in either zero or two or more non-empty fields, the behavior is unspecified. If it results in one non-empty field, that field is taken as the pathname.

If the pathname does not begin with a '/' it shall be treated as relative to the current working directory of the process, not relative to the directory containing the makefile. If the file does not exist in this location, it is unspecified whether additional directories are searched.

The contents of the file specified by the pathname shall be read and processed as if they appeared in the makefile in place of the include line. If the file ends with an escaped <newline> the behavior is unspecified.

The file may itself contain further include lines. Implementations shall support nesting of include files up to a depth of at least 16.

Makefile Execution

Makefile command lines shall be processed one at a time.

Makefile command lines can have one or more of the following prefixes: a <hyphen> (' - '), an at-sign (' @ '), or a <plus-sign> (' + '). These shall modify the way in which *make* processes the command.

- If the command prefix contains a <hyphen>, or the **-i** option is present, or the special target **.IGNORE** has either the current target as a prerequisite or has no prerequisites, any error found while executing the command shall be ignored.
- @ If the command prefix contains an at-sign and the *make* utility command line **-n** option is not specified, or the **-s** option is present, or the special target **.SILENT** has either the current target as a prerequisite or has no prerequisites, the command shall not be written to standard output before it is executed.

- + If the command prefix contains a <plus-sign>, this indicates a makefile command line that shall be executed even if **-n**, **-q**, or **-t** is specified.

An *execution line* is built from the command line by removing any prefix characters. Except as described under the at-sign prefix, the execution line shall be written to the standard output, optionally preceded by a <tab>. The execution line shall then be executed by a shell as if it were passed as the argument to the *system()* interface, except that the shell **-e** option shall also be in effect. The environment for the command being executed shall contain all of the variables in the environment of *make*.

By default, when *make* receives a non-zero status from the execution of a command, it shall terminate with an error message to standard error.

Target Rules

Target rules are formatted as follows:

```
target [target...]: [prerequisite...] [; command]
[<tab>command
<tab>command
...]
```

line that does not begin with <tab>

Target entries are specified by a <blank>-separated, non-null list of targets, then a <colon>, then a <blank>-separated, possibly empty list of prerequisites. Text following a <semicolon>, if any, and all following lines that begin with a <tab>, are makefile command lines to be executed to update the target. The first non-empty line that does not begin with a <tab> or ' #' shall begin a new entry. An empty or blank line, or a line beginning with ' #', may begin a new entry.

Applications shall select target names from the set of characters consisting solely of periods, underscores, digits, and alphabetic characters from the portable character set (see XBD Section 6.1, on page 125). Implementations may allow other characters in target names as extensions. The interpretation of targets containing the characters ' %' and ' "' is implementation-defined.

A target that has prerequisites, but does not have any commands, can be used to add to the prerequisite list for that target. Only one target rule for any given target can contain commands.

Lines that begin with one of the following are called *special targets* and control the operation of *make*:

.DEFAULT If the makefile uses this special target, the application shall ensure that it is specified with commands, but without prerequisites. The commands shall be used by *make* if there are no other rules available to build a target.

.IGNORE Prerequisites of this special target are targets themselves; this shall cause errors from commands associated with them to be ignored in the same manner as specified by the **-i** option. Subsequent occurrences of **.IGNORE** shall add to the list of targets ignoring command errors. If no prerequisites are specified, *make* shall behave as if the **-i** option had been specified and errors from all commands associated with all targets shall be ignored.

.POSIX The application shall ensure that this special target is specified without prerequisites or commands. If it appears as the first non-comment line in the makefile, *make* shall process the makefile as specified by this section; otherwise, the behavior of *make* is unspecified.

95744 **.PRECIOUS** Prerequisites of this special target shall not be removed if *make* receives one of the
 95745 asynchronous events explicitly described in the ASYNCHRONOUS EVENTS
 95746 section. Subsequent occurrences of **.PRECIOUS** shall add to the list of precious
 95747 files. If no prerequisites are specified, all targets in the makefile shall be treated as
 95748 if specified with **.PRECIOUS**.

95749 XSI **.SCCS_GET** The application shall ensure that this special target is specified without
 95750 prerequisites. If this special target is included in a makefile, the commands
 95751 specified with this target shall replace the default commands associated with this
 95752 special target (see [Default Rules](#), on page 2919). The commands specified with this
 95753 target are used to get all SCCS files that are not found in the current directory.
 95754 When source files are named in a dependency list, *make* shall treat them just like
 95755 any other target. Because the source file is presumed to be present in the directory,
 95756 there is no need to add an entry for it to the makefile. When a target has no
 95757 dependencies, but is present in the directory, *make* shall assume that that file is up-
 95758 to-date. If, however, an SCCS file named **SCCS/s.source_file** is found for a target
 95759 *source_file*, *make* compares the timestamp of the target file with that of the
 95760 **SCCS/s.source_file** to ensure the target is up-to-date. If the target is missing, or if
 95761 the SCCS file is newer, *make* shall automatically issue the commands specified for
 95762 the **.SCCS_GET** special target to retrieve the most recent version. However, if the
 95763 target is writable by anyone, *make* shall not retrieve a new version.

95764 **.SILENT** Prerequisites of this special target are targets themselves; this shall cause
 95765 commands associated with them not to be written to the standard output before
 95766 they are executed. Subsequent occurrences of **.SILENT** shall add to the list of
 95767 targets with silent commands. If no prerequisites are specified, *make* shall behave
 95768 as if the **-s** option had been specified and no commands or touch messages
 95769 associated with any target shall be written to standard output.

95770 **.SUFFIXES** Prerequisites of **.SUFFIXES** shall be appended to the list of known suffixes and are
 95771 used in conjunction with the inference rules (see [Inference Rules](#), on page 2916). If
 95772 **.SUFFIXES** does not have any prerequisites, the list of known suffixes shall be
 95773 cleared.

95774 The special targets **.IGNORE**, **.POSIX**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES** shall be specified
 95775 without commands.

95776 Targets with names consisting of a leading <period> followed by the uppercase letters "POSIX"
 95777 and then any other characters are reserved for future standardization. Targets with names
 95778 consisting of a leading <period> followed by one or more uppercase letters are reserved for
 95779 implementation extensions.

95780 **Macros**

95781 Macro definitions are in the form:

95782 *string1* = [*string2*]

95783 The macro named *string1* is defined as having the value of *string2*, where *string2* is defined as all
 95784 characters, if any, after the <equals-sign>, up to a comment character (' #') or an unescaped
 95785 <newline>. Any <blank> characters immediately before or after the <equals-sign> shall be
 95786 ignored.

95787 Applications shall select macro names from the set of characters consisting solely of periods,
 95788 underscores, digits, and alphabetics from the portable character set (see [XBD Section 6.1](#), on
 95789 page 125). A macro name shall not contain an <equals-sign>. Implementations may allow other

characters in macro names as extensions.

Macros can appear anywhere in the makefile. Macro expansions using the forms $\$(string1)$ or $\${string1}$ shall be replaced by *string2*, as follows:

- Macros in target lines shall be evaluated when the target line is read.
- Macros in makefile command lines shall be evaluated when the command is executed.
- Macros in the string before the <equals-sign> in a macro definition shall be evaluated when the macro assignment is made.
- Macros after the <equals-sign> in a macro definition shall not be evaluated until the defined macro is used in a rule or command, or before the <equals-sign> in a macro definition.

The parentheses or braces are optional if *string1* is a single character. The macro $\$\$$ shall be replaced by the single character '\$'. If *string1* in a macro expansion contains a macro expansion, the results are unspecified.

Macro expansions using the forms $\$(string1[:subst1=[subst2]])$ or $\${string1[:subst1=[subst2]]}$ can be used to replace all occurrences of *subst1* with *subst2* when the macro substitution is performed. The *subst1* to be replaced shall be recognized when it is a suffix at the end of a word in *string1* (where a *word*, in this context, is defined to be a string delimited by the beginning of the line, a <blank>, or a <newline>). If *string1* in a macro expansion contains a macro expansion, the results are unspecified.

Macro expansions in *string1* of macro definition lines shall be evaluated when read. Macro expansions in *string2* of macro definition lines shall be performed when the macro identified by *string1* is expanded in a rule or command.

Macro definitions shall be taken from the following sources, in the following logical order, before the makefile(s) are read.

1. Macros specified on the *make* utility command line, in the order specified on the command line. It is unspecified whether the internal macros defined in [Internal Macros](#) (on page 2917) are accepted from this source.
2. Macros defined by the *MAKEFLAGS* environment variable, in the order specified in the environment variable. It is unspecified whether the internal macros defined in [Internal Macros](#) (on page 2917) are accepted from this source.
3. The contents of the environment, excluding the *MAKEFLAGS* and *SHELL* variables and including the variables with null values.
4. Macros defined in the inference rules built into *make*.

Macro definitions from these sources shall not override macro definitions from a lower-numbered source. Macro definitions from a single source (for example, the *make* utility command line, the *MAKEFLAGS* environment variable, or the other environment variables) shall override previous macro definitions from the same source.

Macros defined in the makefile(s) shall override macro definitions that occur before them in the makefile(s) and macro definitions from source 4. If the *-e* option is not specified, macros defined in the makefile(s) shall override macro definitions from source 3. Macros defined in the makefile(s) shall not override macro definitions from source 1 or source 2.

Before the makefile(s) are read, all of the *make* utility command line options (except *-f* and *-p*) and *make* utility command line macro definitions (except any for the *MAKEFLAGS* macro), not already included in the *MAKEFLAGS* macro, shall be added to the *MAKEFLAGS* macro, quoted

in an implementation-defined manner such that when *MAKEFLAGS* is read by another instance of the *make* command, the original macro's value is recovered. Other implementation-defined options and macros may also be added to the *MAKEFLAGS* macro. If this modifies the value of the *MAKEFLAGS* macro, or, if the *MAKEFLAGS* macro is modified at any subsequent time, the *MAKEFLAGS* environment variable shall be modified to match the new value of the *MAKEFLAGS* macro. The result of setting *MAKEFLAGS* in the Makefile is unspecified.

Before the makefile(s) are read, all of the *make* utility command line macro definitions (except the *MAKEFLAGS* macro or the *SHELL* macro) shall be added to the environment of *make*. Other implementation-defined variables may also be added to the environment of *make*.

The **SHELL** macro shall be treated specially. It shall be provided by *make* and set to the pathname of the shell command language interpreter (see *sh*). The *SHELL* environment variable shall not affect the value of the **SHELL** macro. If **SHELL** is defined in the makefile or is specified on the command line, it shall replace the original value of the **SHELL** macro, but shall not affect the *SHELL* environment variable. Other effects of defining **SHELL** in the makefile or on the command line are implementation-defined.

Inference Rules

Inference rules are formatted as follows:

```
target:
<tab>command
[<tab>command]
...
line that does not begin with <tab> or #
```

The application shall ensure that the *target* portion is a valid target name (see [Target Rules](#), on page 2913) of the form *.s2* or *.s1.s2* (where *.s1* and *.s2* are suffixes that have been given as prerequisites of the **.SUFFIXES** special target and *s1* and *s2* do not contain any <slash> or <period> characters.) If there is only one <period> in the target, it is a single-suffix inference rule. Targets with two periods are double-suffix inference rules. Inference rules can have only one target before the <colon>.

The application shall ensure that the makefile does not specify prerequisites for inference rules; no characters other than white space shall follow the <colon> in the first line, except when creating the *empty rule*, described below. Prerequisites are inferred, as described below.

Inference rules can be redefined. A target that matches an existing inference rule shall overwrite the old inference rule. An empty rule can be created with a command consisting of simply a <semicolon> (that is, the rule still exists and is found during inference rule search, but since it is empty, execution has no effect). The empty rule can also be formatted as follows:

```
rule: ;
```

where zero or more <blank> characters separate the <colon> and <semicolon>.

The *make* utility uses the suffixes of targets and their prerequisites to infer how a target can be made up-to-date. A list of inference rules defines the commands to be executed. By default, *make* contains a built-in set of inference rules. Additional rules can be specified in the makefile.

The special target **.SUFFIXES** contains as its prerequisites a list of suffixes that shall be used by the inference rules. The order in which the suffixes are specified defines the order in which the inference rules for the suffixes are used. New suffixes shall be appended to the current list by specifying a **.SUFFIXES** special target in the makefile. A **.SUFFIXES** target with no prerequisites shall clear the list of suffixes. An empty **.SUFFIXES** target followed by a new **.SUFFIXES** list is

required to change the order of the suffixes.

Normally, the user would provide an inference rule for each suffix. The inference rule to update a target with a suffix **.s1** from a prerequisite with a suffix **.s2** is specified as a target **.s2.s1**. The internal macros provide the means to specify general inference rules (see [Internal Macros](#)).

When no target rule is found to update a target, the inference rules shall be checked. The suffix of the target (**.s1**) to be built is compared to the list of suffixes specified by the **.SUFFIXES** special targets. If the **.s1** suffix is found in **.SUFFIXES**, the inference rules shall be searched in the order defined for the first **.s2.s1** rule whose prerequisite file (**\$.s2**) exists. If the target is out-of-date with respect to this prerequisite, the commands for that inference rule shall be executed.

If the target to be built does not contain a suffix and there is no rule for the target, the single suffix inference rules shall be checked. The single-suffix inference rules define how to build a target if a file is found with a name that matches the target name with one of the single suffixes appended. A rule with one suffix **.s2** is the definition of how to build *target* from **target.s2**. The other suffix (**.s1**) is treated as null.

A `<tilde> (' ~')` in the above rules refers to an SCCS file in the current directory. Thus, the rule **.c~.o** would transform an SCCS C-language source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the `' ~'` is a way of changing any file reference into an SCCS file reference.

Libraries

If a target or prerequisite contains parentheses, it shall be treated as a member of an archive library. For the *lib(member.o)* expression *lib* refers to the name of the archive library and *member.o* to the member name. The application shall ensure that the member is an object file with the **.o** suffix. The modification time of the expression is the modification time for the member as kept in the archive library; see [ar](#). The **.a** suffix shall refer to an archive library. The **.s2.a** rule shall be used to update a member in the library from a file with a suffix **.s2**.

Internal Macros

The *make* utility shall maintain five internal macros that can be used in target and inference rules. In order to clearly define the meaning of these macros, some clarification of the terms *target rule*, *inference rule*, *target*, and *prerequisite* is necessary.

Target rules are specified by the user in a makefile for a particular target. Inference rules are user-specified or *make*-specified rules for a particular class of target name. Explicit prerequisites are those prerequisites specified in a makefile on target lines. Implicit prerequisites are those prerequisites that are generated when inference rules are used. Inference rules are applied to implicit prerequisites or to explicit prerequisites that do not have target rules defined for them in the makefile. Target rules are applied to targets specified in the makefile.

Before any target in the makefile is updated, each of its prerequisites (both explicit and implicit) shall be updated. This shall be accomplished by recursively processing each prerequisite. Upon recursion, each prerequisite shall become a target itself. Its prerequisites in turn shall be processed recursively until a target is found that has no prerequisites, at which point the recursion stops. The recursion shall then back up, updating each target as it goes.

In the definitions that follow, the word *target* refers to one of:

- A target specified in the makefile

make*Utilities*

- 95921 • An explicit prerequisite specified in the makefile that becomes the target when *make*
 95922 processes it during recursion
- 95923 • An implicit prerequisite that becomes a target when *make* processes it during recursion
- 95924 In the definitions that follow, the word *prerequisite* refers to one of the following:
- 95925 • An explicit prerequisite specified in the makefile for a particular target
- 95926 • An implicit prerequisite generated as a result of locating an appropriate inference rule and
 95927 corresponding file that matches the suffix of the target
- 95928 The five internal macros are:
- 95929 **\$@** The **\$@** shall evaluate to the full target name of the current target, or the archive
 95930 filename part of a library archive target. It shall be evaluated for both target and
 95931 inference rules.
- 95932 For example, in the **.c.a** inference rule, **\$@** represents the out-of-date **.a** file to be built.
 95933 Similarly, in a makefile target rule to build **lib.a** from **file.c**, **\$@** represents the out-of-
 95934 date **lib.a**.
- 95935 **\$%** The **\$%** macro shall be evaluated only when the current target is an archive library
 95936 member of the form *libname(member.o)*. In these cases, **\$@** shall evaluate to *libname* and
 95937 **\$%** shall evaluate to *member.o*. The **\$%** macro shall be evaluated for both target and
 95938 inference rules.
- 95939 For example, in a makefile target rule to build **lib.a(file.o)**, **\$%** represents **file.o**, as
 95940 opposed to **\$@**, which represents **lib.a**.
- 95941 **\$?** The **\$?** macro shall evaluate to the list of prerequisites that are newer than the current
 95942 target. It shall be evaluated for both target and inference rules.
- 95943 For example, in a makefile target rule to build *prog* from **file1.o**, **file2.o**, and **file3.o**, and
 95944 where *prog* is not out-of-date with respect to **file1.o**, but is out-of-date with respect to
 95945 **file2.o** and **file3.o**, **\$?** represents **file2.o** and **file3.o**.
- 95946 **\$<** In an inference rule, the **\$<** macro shall evaluate to the filename whose existence
 95947 allowed the inference rule to be chosen for the target. In the **.DEFAULT** rule, the **\$<**
 95948 macro shall evaluate to the current target name. The meaning of the **\$<** macro shall be
 95949 otherwise unspecified.
- 95950 For example, in the **.c.a** inference rule, **\$<** represents the prerequisite **.c** file.
- 95951 **\$*** The **\$*** macro shall evaluate to the current target name with its suffix deleted. It shall be
 95952 evaluated at least for inference rules.
- 95953 For example, in the **.c.a** inference rule, **\$*.o** represents the out-of-date **.o** file that
 95954 corresponds to the prerequisite **.c** file.
- 95955 Each of the internal macros has an alternative form. When an uppercase 'D' or 'F' is appended
 95956 to any of the macros, the meaning shall be changed to the *directory part* for 'D' and *filename part*
 95957 for 'F'. The directory part is the path prefix of the file without a trailing <slash>; for the current
 95958 directory, the directory part is ' '. When the **\$?** macro contains more than one prerequisite
 95959 filename, the **\$(?D)** and **\$(?F)** (or **\${?D}** and **\${?F}**) macros expand to a list of directory name parts
 95960 and filename parts respectively.
- 95961 For the target *lib(member.o)* and the **s2.a** rule, the internal macros shall be defined as:

Utilities

make

95962 \$< *member.s2*
 95963 \$* *member*
 95964 \$@ *lib*
 95965 \$? *member.s2*
 95966 \$% *member.o*

95967 **Default Rules**

95968 The default rules for *make* shall achieve results that are the same as if the following were used.
 95969 Implementations that do not support the C-Language Development Utilities option may omit
 95970 **CC**, **CFLAGS**, **YACC**, **YFLAGS**, **LEX**, **LFLAGS**, **LDFLAGS**, and the **.c**, **.y**, and **.l** inference rules.
 95971 Implementations that do not support FORTRAN may omit **FC**, **FFLAGS**, and the **.f** inference
 95972 rules. Implementations may provide additional macros and rules.

95973 *SPECIAL TARGETS*

95974 XSI **.SCCS_GET:** *sccs \$(SCCSFLAGS) get \$(SCCSGETFLAGS) \$@*

95975 XSI **.SUFFIXES:** *.o .c .y .l .a .sh .f .c~ .y~ .l~ .sh~ .f~*

95976 **MACROS**

95977 MAKE=make
 95978 AR=ar
 95979 ARFLAGS=-rv
 95980 YACC=yacc
 95981 YFLAGS=
 95982 LEX=lex
 95983 LFLAGS=
 95984 LDFLAGS=
 95985 CC=c99
 95986 CFLAGS=-O
 95987 FC=fort77
 95988 FFLAGS=-O 1

95989 XSI GET=get
 95990 GFLAGS=
 95991 SCCSFLAGS=
 95992 SCCSGETFLAGS=-s

95993 *SINGLE SUFFIX RULES*

95994 **.c:**
 95995 \$(CC) \$(CFLAGS) \$(LDFLAGS) -o \$@ \$<

95996 **.f:**
 95997 \$(FC) \$(FFLAGS) \$(LDFLAGS) -o \$@ \$<

95998 **.sh:**
 95999 cp \$< \$@
 96000 chmod a+x \$@

96001 XSI **.c~:**
 96002 \$(GET) \$(GFLAGS) -p \$< > \$*.c

make*Utilities*

```

96003      $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c
96004  .f~:
96005      $(GET) $(GFLAGS) -p $< > $*.f
96006      $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.f
96007  .sh~:
96008      $(GET) $(GFLAGS) -p $< > $*.sh
96009      cp $*.sh $@
96010      chmod a+x $@

```

96011 *DOUBLE SUFFIX RULES*

```

96012  .c.o:
96013      $(CC) $(CFLAGS) -c $<
96014  .f.o:
96015      $(FC) $(FFLAGS) -c $<
96016  .y.o:
96017      $(YACC) $(YFLAGS) $<
96018      $(CC) $(CFLAGS) -c y.tab.c
96019      rm -f y.tab.c
96020      mv y.tab.o $@
96021  .l.o:
96022      $(LEX) $(LFLAGS) $<
96023      $(CC) $(CFLAGS) -c lex.yy.c
96024      rm -f lex.yy.c
96025      mv lex.yy.o $@
96026  .y.c:
96027      $(YACC) $(YFLAGS) $<
96028      mv y.tab.c $@
96029  .l.c:
96030      $(LEX) $(LFLAGS) $<
96031      mv lex.yy.c $@

```

```

96032  XSI .c~.o:
96033      $(GET) $(GFLAGS) -p $< > $*.c
96034      $(CC) $(CFLAGS) -c $*.c
96035  .f~.o:
96036      $(GET) $(GFLAGS) -p $< > $*.f
96037      $(FC) $(FFLAGS) -c $*.f
96038  .y~.o:
96039      $(GET) $(GFLAGS) -p $< > $*.y
96040      $(YACC) $(YFLAGS) $*.y
96041      $(CC) $(CFLAGS) -c y.tab.c
96042      rm -f y.tab.c
96043      mv y.tab.o $@
96044  .l~.o:
96045      $(GET) $(GFLAGS) -p $< > $*.l
96046      $(LEX) $(LFLAGS) $*.l

```

Utilities

make

```

96047      $(CC) $(CFLAGS) -c lex.yy.c
96048      rm -f lex.yy.c
96049      mv lex.yy.o $@

96050  .y~.c:
96051      $(GET) $(GFLAGS) -p $< > $*.y
96052      $(YACC) $(YFLAGS) $*.y
96053      mv y.tab.c $@

96054  .l~.c:
96055      $(GET) $(GFLAGS) -p $< > $*.l
96056      $(LEX) $(LFLAGS) $*.l
96057      mv lex.yy.c $@

```

```

96058  .c.a:
96059      $(CC) -c $(CFLAGS) $<
96060      $(AR) $(ARFLAGS) $@ $*.o
96061      rm -f $*.o

96062  .f.a:
96063      $(FC) -c $(FFLAGS) $<
96064      $(AR) $(ARFLAGS) $@ $*.o
96065      rm -f $*.o

```

EXIT STATUS

When the **-q** option is specified, the *make* utility shall exit with one of the following values:

- 0 Successful completion.
- 1 The target was not up-to-date.
- >1 An error occurred.

When the **-q** option is not specified, the *make* utility shall exit with one of the following values:

- 0 Successful completion.
- >0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

If there is a source file (such as *./source.c*) and there are two SCCS files corresponding to it (*./s.source.c* and *./SCCS/s.source.c*), on XSI-conformant systems *make* uses the SCCS file in the current directory. However, users are advised to use the underlying SCCS utilities (*admin*, *delta*, *get*, and so on) or the *sccs* utility for all source files in a given directory. If both forms are used for a given source file, future developers are very likely to be confused.

It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to guarantee that they are not affected by local extensions.

The **-k** and **-S** options are both present so that the relationship between the command line, the *MAKEFLAGS* variable, and the makefile can be controlled precisely. If the **k** flag is passed in *MAKEFLAGS* and a command is of the form:

```
$(MAKE) -S foo
```

then the default behavior is restored for the child *make*.

When the `-n` option is specified, it is always added to `MAKEFLAGS`. This allows a recursive `make -n target` to be used to see all of the action that would be taken to update *target*.

Because of widespread historical practice, interpreting a `<number-sign>` (`'#'`) inside a variable as the start of a comment has the unfortunate side-effect of making it impossible to place a `<number-sign>` in a variable, thus forbidding something like:

```
CFLAGS = "-D COMMENT_CHAR='#'"
```

Many historical *make* utilities stop chaining together inference rules when an intermediate target is nonexistent. For example, it might be possible for a *make* to determine that both `.yc` and `.co` could be used to convert a `.y` to a `.o`. Instead, in this case, *make* requires the use of a `.yo` rule.

The best way to provide portable makefiles is to include all of the rules needed in the makefile itself. The rules provided use only features provided by other parts of this volume of POSIX.1-2008. The default rules include rules for optional commands in this volume of POSIX.1-2008. Only rules pertaining to commands that are provided are needed in an implementation's default set.

Macros used within other macros are evaluated when the new macro is used rather than when the new macro is defined. Therefore:

```
MACRO = value1
NEW    = $(MACRO)
MACRO = value2
```

```
target:
    echo $(NEW)
```

would produce *value2* and not *value1* since `NEW` was not expanded until it was needed in the `echo` command line.

Some historical applications have been known to intermix *target_name* and *macro=name* operands on the command line, expecting that all of the macros are processed before any of the targets are dealt with. Conforming applications do not do this, although some backwards-compatibility support may be included in some implementations.

The following characters in filenames may give trouble: `'=`, `':`, `'`, `'`, `'`, `'`, single-quote, and `'@`. In include filenames, pattern matching characters and `'` should also be avoided, as they may be treated as special by some implementations.

For inference rules, the description of `$<` and `$?` seem similar. However, an example shows the minor difference. In a makefile containing:

```
foo.o: foo.h
```

if `foo.h` is newer than `foo.o`, yet `foo.c` is older than `foo.o`, the built-in rule to make `foo.o` from `foo.c` is used, with `$<` equal to `foo.c` and `$?` equal to `foo.h`. If `foo.c` is also newer than `foo.o`, `$<` is equal to `foo.c` and `$?` is equal to `foo.h`.

EXAMPLES

1. The following command:

```
make
```

makes the first target found in the makefile.

2. The following command:

```
make junk
```

- 96131 makes the target **junk**.
- 96132 3. The following makefile says that **pgm** depends on two files, **a.o** and **b.o**, and that they in
- 96133 turn depend on their corresponding source files (**a.c** and **b.c**), and a common file **incl.h**:
- 96134 `pgm: a.o b.o`
- 96135 `c99 a.o b.o -o pgm`
- 96136 `a.o: incl.h a.c`
- 96137 `c99 -c a.c`
- 96138 `b.o: incl.h b.c`
- 96139 `c99 -c b.c`
- 96140 4. An example for making optimized **.o** files from **.c** files is:
- 96141 `.c.o:`
- 96142 `c99 -c -O $*.c`
- 96143 or:
- 96144 `.c.o:`
- 96145 `c99 -c -O $<`
- 96146 5. The most common use of the archive interface follows. Here, it is assumed that the source
- 96147 files are all C-language source:
- 96148 `lib: lib(file1.o) lib(file2.o) lib(file3.o)`
- 96149 `@echo lib is now up-to-date`
- 96150 The **.c.a** rule is used to make **file1.o**, **file2.o**, and **file3.o** and insert them into **lib**.
- 96151 The treatment of escaped <newline> characters throughout the makefile is historical
- 96152 practice. For example, the inference rule:
- 96153 `.c.o\`
- 96154 `:`
- 96155 works, and the macro:
- 96156 `f= bar baz\`
- 96157 `biz`
- 96158 `a:`
- 96159 `echo ==$f==`
- 96160 `echoes "==bar baz biz=="`.
- 96161 If **\$?** were:
- 96162 `/usr/include/stdio.h /usr/include/unistd.h foo.h`
- 96163 then **\$(?D)** would be:
- 96164 `/usr/include /usr/include .`
- 96165 and **\$(?F)** would be:
- 96166 `stdio.h unistd.h foo.h`
- 96167 6. The contents of the built-in rules can be viewed by running:
- 96168 `make -p -f /dev/null 2>/dev/null`

RATIONALE

The *make* utility described in this volume of POSIX.1-2008 is intended to provide the means for changing portable source code into executables that can be run on an POSIX.1-2008-conforming system. It reflects the most common features present in System V and BSD *makes*.

Historically, the *make* utility has been an especially fertile ground for vendor and research organization-specific syntax modifications and extensions. Examples include:

- Syntax supporting parallel execution (such as from various multi-processor vendors, GNU, and others)
- Additional “operators” separating targets and their prerequisites (System V, BSD, and others)
- Specifying that command lines containing the strings “\${MAKE}” and “\$(MAKE)” are executed when the `-n` option is specified (GNU and System V)
- Modifications of the meaning of internal macros when referencing libraries (BSD and others)
- Using a single instance of the shell for all of the command lines of the target (BSD and others)
- Allowing <space> characters as well as <tab> characters to delimit command lines (BSD)
- Adding C preprocessor-style “include” and “ifdef” constructs (System V, GNU, BSD, and others)
- Remote execution of command lines (Sprite and others)
- Specifying additional special targets (BSD, System V, and most others)

Additionally, many vendors and research organizations have rethought the basic concepts of *make*, creating vastly extended, as well as completely new, syntaxes. Each of these versions of *make* fulfills the needs of a different community of users; it is unreasonable for this volume of POSIX.1-2008 to require behavior that would be incompatible (and probably inferior) to historical practice for such a community.

In similar circumstances, when the industry has enough sufficiently incompatible formats as to make them irreconcilable, this volume of POSIX.1-2008 has followed one or both of two courses of action. Commands have been renamed (*cksum*, *echo*, and *pax*) and/or command line options have been provided to select the desired behavior (*grep*, *od*, and *pax*).

Because the syntax specified for the *make* utility is, by and large, a subset of the syntaxes accepted by almost all versions of *make*, it was decided that it would be counter-productive to change the name. And since the makefile itself is a basic unit of portability, it would not be completely effective to reserve a new option letter, such as *make -P*, to achieve the portable behavior. Therefore, the special target **.POSIX** was added to the makefile, allowing users to specify “standard” behavior. This special target does not preclude extensions in the *make* utility, nor does it preclude such extensions being used by the makefile specifying the target; it does, however, preclude any extensions from being applied that could alter the behavior of previously valid syntax; such extensions must be controlled via command line options or new special targets. It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to guarantee that they are not affected by local extensions.

The portable version of *make* described in this reference page is not intended to be the state-of-the-art software generation tool and, as such, some newer and more leading-edge features have not been included. An attempt has been made to describe the portable makefile in a manner that does not preclude such extensions as long as they do not disturb the portable behavior described

here.

When the **-n** option is specified, it is always added to **MAKEFLAGS**. This allows a recursive **make -n target** to be used to see all of the action that would be taken to update *target*.

The definition of **MAKEFLAGS** allows both the System V letter string and the BSD command line formats. The two formats are sufficiently different to allow implementations to support both without ambiguity.

Early proposals stated that an “unquoted” <number-sign> was treated as the start of a comment. The *make* utility does not pay any attention to quotes. A <number-sign> starts a comment regardless of its surroundings.

The text about “other implementation-defined pathnames may also be tried” in addition to **./makefile** and **./Makefile** is to allow such extensions as **SCCS/s.Makefile** and other variations. It was made an implementation-defined requirement (as opposed to unspecified behavior) to highlight surprising implementations that might select something unexpected like **/etc/Makefile**. XSI-conformant systems also try **./s.makefile**, **SCCS/s.makefile**, **./s.Makefile**, and **SCCS/s.Makefile**.

Early proposals contained the macro **NPROC** as a means of specifying that *make* should use *n* processes to do the work required. While this feature is a valuable extension for many systems, it is not common usage and could require other non-trivial extensions to makefile syntax. This extension is not required by this volume of POSIX.1-2008, but could be provided as a compatible extension. The macro **PARALLEL** is used by some historical systems with essentially the same meaning (but without using a name that is a common system limit value). It is suggested that implementors recognize the existing use of **NPROC** and/or **PARALLEL** as extensions to *make*.

The default rules are based on System V. The default **CC=** value is *c99* instead of *cc* because this volume of POSIX.1-2008 does not standardize the utility named *cc*. Thus, every conforming application would be required to define **CC=c99** to expect to run. There is no advantage conferred by the hope that the makefile might hit the “preferred” compiler because this cannot be guaranteed to work. Also, since the portable makescript can only use the *c99* options, no advantage is conferred in terms of what the script can do. It is a quality-of-implementation issue as to whether *c99* is as valuable as *cc*.

The **-d** option to *make* is frequently used to produce debugging information, but is too implementation-defined to add to this volume of POSIX.1-2008.

The **-p** option is not passed in **MAKEFLAGS** on most historical implementations and to change this would cause many implementations to break without sufficiently increased portability.

Commands that begin with a <plus-sign> ('+') are executed even if the **-n** option is present. Based on the GNU version of *make*, the behavior of **-n** when the <plus-sign> prefix is encountered has been extended to apply to **-q** and **-t** as well. However, the System V convention of forcing command execution with **-n** when the command line of a target contains either of the strings **"\$(MAKE)"** or **"\${MAKE}"** has not been adopted. This functionality appeared in early proposals, but the danger of this approach was pointed out with the following example of a portion of a makefile:

```
subdir:
    cd subdir; rm all_the_files; $(MAKE)
```

The loss of the System V behavior in this case is well-balanced by the safety afforded to other makefiles that were not aware of this situation. In any event, the command line <plus-sign> prefix can provide the desired functionality.

The double <colon> in the target rule format is supported in BSD systems to allow more than

one target line containing the same target name to have commands associated with it. Since this is not functionality described in the SVID or XPG3 it has been allowed as an extension, but not mandated.

The default rules are provided with text specifying that the built-in rules shall be the same as if the listed set were used. The intent is that implementations should be able to use the rules without change, but will be allowed to alter them in ways that do not affect the primary behavior.

The best way to provide portable makefiles is to include all of the rules needed in the makefile itself. The rules provided use only features provided by other portions of this volume of POSIX.1-2008. The default rules include rules for optional commands in this volume of POSIX.1-2008. Only rules pertaining to commands that are provided are needed in the default set of an implementation.

One point of discussion was whether to drop the default rules list from this volume of POSIX.1-2008. They provide convenience, but do not enhance portability of applications. The prime benefit is in portability of users who wish to type *make command* and have the command build from a **command.c** file.

The historical *MAKESHELL* feature was omitted. In some implementations it is used to let a user override the shell to be used to run *make* commands. This was confusing; for a portable *make*, the shell should be chosen by the makefile writer or specified on the *make* command line and not by a user running *make*.

The *make* utilities in most historical implementations process the prerequisites of a target in left-to-right order, and the makefile format requires this. It supports the standard idiom used in many makefiles that produce *yacc* programs; for example:

```
foo: y.tab.o lex.o main.o
    $(CC) $(CFLAGS) -o $@ t.y.tab.o lex.o main.o
```

In this example, if *make* chose any arbitrary order, the **lex.o** might not be made with the correct **y.tab.h**. Although there may be better ways to express this relationship, it is widely used historically. Implementations that desire to update prerequisites in parallel should require an explicit extension to *make* or the makefile format to accomplish it, as described previously.

The algorithm for determining a new entry for target rules is partially unspecified. Some historical *makes* allow blank, empty, or comment lines within the collection of commands marked by leading <tab> characters. A conforming makefile must ensure that each command starts with a <tab>, but implementations are free to ignore blank, empty, and comment lines without triggering the start of a new entry.

The ASYNCHRONOUS EVENTS section includes having SIGTERM and SIGHUP, along with the more traditional SIGINT and SIGQUIT, remove the current target unless directed not to do so. SIGTERM and SIGHUP were added to parallel other utilities that have historically cleaned up their work as a result of these signals. When *make* receives any signal other than SIGQUIT, it is required to resend itself the signal it received so that it exits with a status that reflects the signal. The results from SIGQUIT are partially unspecified because, on systems that create **core** files upon receipt of SIGQUIT, the **core** from *make* would conflict with a **core** file from the command that was running when the SIGQUIT arrived. The main concern was to prevent damaged files from appearing up-to-date when *make* is rerun.

The **.PRECIOUS** special target was extended to affect all targets globally (by specifying no prerequisites). The **.IGNORE** and **.SILENT** special targets were extended to allow prerequisites; it was judged to be more useful in some cases to be able to turn off errors or echoing for a list of targets than for the entire makefile. These extensions to *make* in System V were made to match

historical practice from the BSD *make*.

Macros are not exported to the environment of commands to be run. This was never the case in any historical *make* and would have serious consequences. The environment is the same as the environment to *make* except that *MAKEFLAGS* and macros defined on the *make* command line are added.

Some implementations do not use *system()* for all command lines, as required by the portable makefile format; as a performance enhancement, they select lines without shell metacharacters for direct execution by *execve()*. There is no requirement that *system()* be used specifically, but merely that the same results be achieved. The metacharacters typically used to bypass the direct *execve()* execution have been any of:

= | ^ () ; & < > * ? [] : \$ \ ' " \ \n

The default in some advanced versions of *make* is to group all the command lines for a target and execute them using a single shell invocation; the System V method is to pass each line individually to a separate shell. The single-shell method has the advantages in performance and the lack of a requirement for many continued lines. However, converting to this newer method has caused portability problems with many historical makefiles, so the behavior with the POSIX makefile is specified to be the same as that of System V. It is suggested that the special target **.ONESHELL** be used as an implementation extension to achieve the single-shell grouping for a target or group of targets.

Novice users of *make* have had difficulty with the historical need to start commands with a <tab>. Since it is often difficult to discern differences between <tab> and <space> characters on terminals or printed listings, confusing bugs can arise. In early proposals, an attempt was made to correct this problem by allowing leading <blank> characters instead of <tab> characters. However, implementors reported many makefiles that failed in subtle ways following this change, and it is difficult to implement a *make* that unambiguously can differentiate between macro and command lines. There is extensive historical practice of allowing leading <space> characters before macro definitions. Forcing macro lines into column 1 would be a significant backwards-compatibility problem for some makefiles. Therefore, historical practice was restored.

There is substantial variation in the handling of include lines by different implementations. However, there is enough commonality for the standard to be able to specify a minimum set of requirements that allow the feature to be used portably. Known variations have been explicitly called out as unspecified behavior in the description.

The System V dynamic dependency feature was not included. It would support:

```
cat: $$e.c
```

that would expand to;

```
cat: cat.c
```

This feature exists only in the new version of System V *make* and, while useful, is not in wide usage. This means that macros are expanded twice for prerequisites: once at makefile parse time and once at target update time.

Consideration was given to adding metarules to the POSIX *make*. This would make **%o: %c** the same as **.c.o:**. This is quite useful and available from some vendors, but it would cause too many changes to this *make* to support. It would have introduced rule chaining and new substitution rules. However, the rules for target names have been set to reserve the **' % '** and **' ' '** characters. These are traditionally used to implement metarules and quoting of target names, respectively. Implementors are strongly encouraged to use these characters only for these purposes.

A request was made to extend the suffix delimiter character from a <period> to any character. The metarules feature in newer *makes* solves this problem in a more general way. This volume of POSIX.1-2008 is staying with the more conservative historical definition.

The standard output format for the **-p** option is not described because it is primarily a debugging option and because the format is not generally useful to programs. In historical implementations the output is not suitable for use in generating makefiles. The **-p** format has been variable across historical implementations. Therefore, the definition of **-p** was only to provide a consistently named option for obtaining *make* script debugging information.

Some historical implementations have not cleared the suffix list with **-r**.

Implementations should be aware that some historical applications have intermixed *target_name* and *macro=value* operands on the command line, expecting that all of the macros are processed before any of the targets are dealt with. Conforming applications do not do this, but some backwards-compatibility support may be warranted.

Empty inference rules are specified with a <semicolon> command rather than omitting all commands, as described in an early proposal. The latter case has no traditional meaning and is reserved for implementation extensions, such as in GNU *make*.

FUTURE DIRECTIONS

None.

SEE ALSO

[Chapter 2](#) (on page 2297), *ar*, *c99*, *get*, *lex*, *sccs*, *sh*, *yacc*

XBD [Section 6.1](#) (on page 125), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

XSH *exec*, *system()*

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the Software Development Utilities option.

The Open Group Corrigendum U029/1 is applied, correcting a typographical error in the SPECIAL TARGETS section.

In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from “otherwise, the home directory of a user of that name is examined” to “otherwise, the value of *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined”.

It is specified whether the command line is related to the makefile or to the *make* command, and the macro processing rules are updated to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term “must” for application requirements.

PASC Interpretation 1003.2 #193 is applied.

Issue 7

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Include lines in makefiles are introduced.

96395
96396

Austin Group Interpretation 1003.1-2001 #131 is applied, changing the **Makefile Execution** section.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

96397 **NAME**

96398 man — display system documentation

96399 **SYNOPSIS**96400 man [-k] *name*...96401 **DESCRIPTION**

96402 The *man* utility shall write information about each of the *name* operands. If *name* is the name of a
 96403 standard utility, *man* at a minimum shall write a message describing the syntax used by the
 96404 standard utility, its options, and operands. If more information is available, the *man* utility shall
 96405 provide it in an implementation-defined manner.

96406 An implementation may provide information for values of *name* other than the standard utilities.
 96407 Standard utilities that are listed as optional and that are not supported by the implementation
 96408 either shall cause a brief message indicating that fact to be displayed or shall cause a full display
 96409 of information as described previously.

96410 **OPTIONS**96411 The *man* utility shall conform to XBD [Section 12.2](#) (on page 215).

96412 The following option shall be supported:

96413 -k Interpret *name* operands as keywords to be used in searching a utilities summary
 96414 database that contains a brief purpose entry for each standard utility and write lines
 96415 from the summary database that match any of the keywords. The keyword search shall
 96416 produce results that are the equivalent of the output of the following command:

```
96417       grep -Ei '
96418       name
96419       name
96420       ...
96421       ' summary-database
```

96422 This assumes that the *summary-database* is a text file with a single entry per line; this
 96423 organization is not required and the example using *grep -Ei* is merely illustrative of the
 96424 type of search intended. The purpose entry to be included in the database shall consist
 96425 of a terse description of the purpose of the utility.

96426 **OPERANDS**

96427 The following operand shall be supported:

96428 *name* A keyword or the name of a standard utility. When -k is not specified and *name*
 96429 does not represent one of the standard utilities, the results are unspecified.

96430 **STDIN**

96431 Not used.

96432 **INPUT FILES**

96433 None.

96434 **ENVIRONMENT VARIABLES**96435 The following environment variables shall affect the execution of *man*:

96436 LANG Provide a default value for the internationalization variables that are unset or null.
 96437 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 96438 variables used to determine the values of locale categories.)

96439 LC_ALL If set to a non-empty string value, override the values of all the other
 96440 internationalization variables.

96441	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and in the summary database). The value of <i>LC_CTYPE</i> need not affect the format of the information written about the <i>name</i> operands.
96442		
96443		
96444		
96445	<i>LC_MESSAGES</i>	
96446		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.
96447		
96448		
96449	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
96450	<i>PAGER</i>	Determine an output filtering command for writing the output to a terminal. Any string acceptable as a <i>command_string</i> operand to the <i>sh -c</i> command shall be valid. When standard output is a terminal device, the reference page output shall be piped through the command. If the <i>PAGER</i> variable is null or not set, the command shall be either <i>more</i> or another paginator utility documented in the system documentation.
96451		
96452		
96453		
96454		
96455		
96456	ASYNCHRONOUS EVENTS	
96457		Default.
96458	STDOUT	
96459		The <i>man</i> utility shall write text describing the syntax of the utility <i>name</i> , its options and its operands, or, when <i>-k</i> is specified, lines from the summary database. The format of this text is implementation-defined.
96460		
96461		
96462	STDERR	
96463		The standard error shall be used for diagnostic messages, and may also be used for informational messages of unspecified format.
96464		
96465	OUTPUT FILES	
96466		None.
96467	EXTENDED DESCRIPTION	
96468		None.
96469	EXIT STATUS	
96470		The following exit values shall be returned:
96471	0	Successful completion.
96472	>0	An error occurred.
96473	CONSEQUENCES OF ERRORS	
96474		Default.
96475	APPLICATION USAGE	
96476		None.
96477	EXAMPLES	
96478		None.
96479	RATIONALE	
96480		It is recognized that the <i>man</i> utility is only of minimal usefulness as specified. The opinion of the standard developers was strongly divided as to how much or how little information <i>man</i> should be required to provide. They considered, however, that the provision of some portable way of accessing documentation would aid user portability. The arguments against a fuller specification were:
96481		
96482		
96483		
96484		

- Large quantities of documentation should not be required on a system that does not have excess disk space.
- The current manual system does not present information in a manner that greatly aids user portability.
- A “better help system” is currently an area in which vendors feel that they can add value to their POSIX implementations.

The `-f` option was considered, but due to implementation differences, it was not included in this volume of POSIX.1-2008.

The description was changed to be more specific about what has to be displayed for a utility. The standard developers considered it insufficient to allow a display of only the synopsis without giving a short description of what each option and operand does.

The “purpose” entry to be included in the database can be similar to the section title (less the numeric prefix) from this volume of POSIX.1-2008 for each utility. These titles are similar to those used in historical systems for this purpose.

See *mailx* for rationale concerning the default paginator.

The caveat in the *LC_CTYPE* description was added because it is not a requirement that an implementation provide reference pages for all of its supported locales on each system; changing *LC_CTYPE* does not necessarily translate the reference page into another language. This is equivalent to the current state of *LC_MESSAGES* in POSIX.1-2008—locale-specific messages are not yet a requirement.

The historical *MANPATH* variable is not included in POSIX because no attempt is made to specify naming conventions for reference page files, nor even to mandate that they are files at all. On some implementations they could be a true database, a hypertext file, or even fixed strings within the *man* executable. The standard developers considered the portability of reference pages to be outside their scope of work. However, users should be aware that *MANPATH* is implemented on a number of historical systems and that it can be used to tailor the search pattern for reference pages from the various categories (utilities, functions, file formats, and so on) when the system administrator reveals the location and conventions for reference pages on the system.

The keyword search can rely on at least the text of the section titles from these utility descriptions, and the implementation may add more keywords. The term “section titles” refers to the strings such as:

```
man - Display system documentation
ps - Report process status
```

FUTURE DIRECTIONS

None.

SEE ALSO

[more](#)

XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

CHANGE HISTORY

First released in Issue 4.

96526 **Issue 5**

96527 The FUTURE DIRECTIONS section is added.

96528 **Issue 7**96529 Austin Group Interpretation 1003.1-2001 #108 is applied, clarifying that informational messages
96530 may appear on standard error.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

mesg

Utilities

96531 **NAME**

96532 mesg — permit or deny messages

96533 **SYNOPSIS**96534 mesg [*y* | *n*]96535 **DESCRIPTION**

96536 The *mesg* utility shall control whether other users are allowed to send messages via *write*, *talk*, or
 96537 other utilities to a terminal device. The terminal device affected shall be determined by searching
 96538 for the first terminal in the sequence of devices associated with standard input, standard output,
 96539 and standard error, respectively. With no arguments, *mesg* shall report the current state without
 96540 changing it. Processes with appropriate privileges may be able to send messages to the terminal
 96541 independent of the current state.

96542 **OPTIONS**

96543 None.

96544 **OPERANDS**

96545 The following operands shall be supported in the POSIX locale:

96546 *y* Grant permission to other users to send messages to the terminal device.96547 *n* Deny permission to other users to send messages to the terminal device.96548 **STDIN**

96549 Not used.

96550 **INPUT FILES**

96551 None.

96552 **ENVIRONMENT VARIABLES**96553 The following environment variables shall affect the execution of *mesg*:

96554 *LANG* Provide a default value for the internationalization variables that are unset or null.
 96555 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 96556 variables used to determine the values of locale categories.)

96557 *LC_ALL* If set to a non-empty string value, override the values of all the other
 96558 internationalization variables.

96559 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 96560 characters (for example, single-byte as opposed to multi-byte characters in
 96561 arguments).

96562 *LC_MESSAGES* Determine the locale that should be used to affect the format and contents of
 96563 diagnostic messages written (by *mesg*) to standard error.
 96564

96565 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

96566 **ASYNCHRONOUS EVENTS**

96567 Default.

96568 **STDOUT**96569 If no operand is specified, *mesg* shall display the current terminal state in an unspecified format.96570 **STDERR**

96571 The standard error shall be used only for diagnostic messages.

96572 **OUTPUT FILES**

96573 None.

96574 **EXTENDED DESCRIPTION**

96575 None.

96576 **EXIT STATUS**

96577 The following exit values shall be returned:

96578 0 Receiving messages is allowed.

96579 1 Receiving messages is not allowed.

96580 >1 An error occurred.

96581 **CONSEQUENCES OF ERRORS**

96582 Default.

96583 **APPLICATION USAGE**

96584 The mechanism by which the message status of the terminal is changed is unspecified.
 96585 Therefore, unspecified actions may cause the status of the terminal to change after *mesg* has
 96586 successfully completed. These actions may include, but are not limited to: another invocation of
 96587 the *mesg* utility, login procedures; invocation of the *stty* utility, invocation of the *chmod* utility or
 96588 *chmod()* function, and so on.

96589 **EXAMPLES**

96590 None.

96591 **RATIONALE**

96592 The terminal changed by *mesg* is that associated with the standard input, output, or error, rather
 96593 than the controlling terminal for the session. This is because users logged in more than once
 96594 should be able to change any of their login terminals without having to stop the job running in
 96595 those sessions. This is not a security problem involving the terminals of other users because
 96596 appropriate privileges would be required to affect the terminal of another user.

96597 The method of checking each of the first three file descriptors in sequence until a terminal is
 96598 found was adopted from System V.

96599 The file */dev/tty* is not specified for the terminal device because it was thought to be too
 96600 restrictive. Typical environment changes for the *n* operand are that write permissions are
 96601 removed for *others* and *group* from the appropriate device. It was decided to leave the actual
 96602 description of what is done as unspecified because of potential differences between
 96603 implementations.

96604 The format for standard output is unspecified because of differences between historical
 96605 implementations. This output is generally not useful to shell scripts (they can use the exit status),
 96606 so exact parsing of the output is unnecessary.

96607 **FUTURE DIRECTIONS**

96608 None.

96609 **SEE ALSO**96610 *talk*, *write*

96611 XBD Chapter 8 (on page 173)

mesg*Utilities*96612 **CHANGE HISTORY**

96613 First released in Issue 2.

96614 **Issue 6**

96615 This utility is marked as part of the User Portability Utilities option.

96616 **Issue 7**96617 The *mesg* utility is moved from the User Portability Utilities option to the Base. User Portability
96618 Utilities is now an option for interactive utilities.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

mkdir**NAME**

mkdir — make directories

SYNOPSIS

mkdir [-p] [-m *mode*] *dir*...

DESCRIPTION

The *mkdir* utility shall create the directories specified by the operands, in the order specified.

For each *dir* operand, the *mkdir* utility shall perform actions equivalent to the *mkdir()* function defined in the System Interfaces volume of POSIX.1-2008, called with the following arguments:

1. The *dir* operand is used as the *path* argument.
2. The value of the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO is used as the *mode* argument. (If the **-m** option is specified, the value of the *mkdir()* *mode* argument is unspecified, but the directory shall at no time have permissions less restrictive than the **-m mode** option-argument.)

OPTIONS

The *mkdir* utility shall conform to XBD [Section 12.2](#) (on page 215).

The following options shall be supported:

-m mode Set the file permission bits of the newly-created directory to the specified *mode* value. The *mode* option-argument shall be the same as the *mode* operand defined for the *chmod* utility. In the *symbolic mode* strings, the *op* characters '+' and '-' shall be interpreted relative to an assumed initial mode of *a=rwx*; '+' shall add permissions to the default mode, '-' shall delete permissions from the default mode.

-p Create any missing intermediate pathname components.

For each *dir* operand that does not name an existing directory, effects equivalent to those caused by the following command shall occur:

```
mkdir -p -m $(umask -S),u+wx $(dirname dir) &&
mkdir [-m mode] dir
```

where the **-m mode** option represents that option supplied to the original invocation of *mkdir*, if any.

Each *dir* operand that names an existing directory shall be ignored without error.

OPERANDS

The following operand shall be supported:

dir A pathname of a directory to be created.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *mkdir*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

mkdir

Utilities

96661	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
96662		
96663	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
96664		
96665		
96666	<i>LC_MESSAGES</i>	
96667		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
96668		
96669	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
96670	ASYNCHRONOUS EVENTS	
96671		Default.
96672	STDOUT	
96673		Not used.
96674	STDERR	
96675		The standard error shall be used only for diagnostic messages.
96676	OUTPUT FILES	
96677		None.
96678	EXTENDED DESCRIPTION	
96679		None.
96680	EXIT STATUS	
96681		The following exit values shall be returned:
96682	0	All the specified directories were created successfully or the -p option was specified and all the specified directories now exist.
96683		
96684	>0	An error occurred.
96685	CONSEQUENCES OF ERRORS	
96686		Default.
96687	APPLICATION USAGE	
96688		The default file mode for directories is <i>a=rwx</i> (777 on most systems) with selected permissions removed in accordance with the file mode creation mask. For intermediate pathname components created by <i>mkdir</i> , the mode is the default modified by <i>u+wx</i> so that the subdirectories can always be created regardless of the file mode creation mask; if different ultimate permissions are desired for the intermediate directories, they can be changed afterwards with <i>chmod</i> .
96689		
96690		
96691		
96692		
96693		
96694		Note that some of the requested directories may have been created even if an error occurs.
96695	EXAMPLES	
96696		None.
96697	RATIONALE	
96698		The System V -m option was included to control the file mode.
96699		The System V -p option was included to create any needed intermediate directories and to complement the functionality provided by <i>rmdir</i> for removing directories in the path prefix as they become empty. Because no error is produced if any path component already exists, the -p option is also useful to ensure that a particular directory exists.
96700		
96701		
96702		
96703		The functionality of <i>mkdir</i> is described substantially through a reference to the <i>mkdir()</i> function

96704 in the System Interfaces volume of POSIX.1-2008. For example, by default, the mode of the
96705 directory is affected by the file mode creation mask in accordance with the specified behavior of
96706 the *mkdir()* function. In this way, there is less duplication of effort required for describing details
96707 of the directory creation.

96708 **FUTURE DIRECTIONS**

96709 None.

96710 **SEE ALSO**

96711 *chmod*, *rm*, *rmdir*, *umask*

96712 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

96713 XSH *mkdir()*

96714 **CHANGE HISTORY**

96715 First released in Issue 2.

96716 **Issue 5**

96717 The FUTURE DIRECTIONS section is added.

96718 **Issue 7**

96719 SD5-XCU-ERN-56 is applied, aligning the *-m* option with the IEEE P1003.2b draft standard to
96720 clarify an ambiguity.

96721 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

mkfifo

Utilities

96722 **NAME**

96723 mkfifo — make FIFO special files

96724 **SYNOPSIS**96725 mkfifo [-m *mode*] *file*...96726 **DESCRIPTION**96727 The *mkfifo* utility shall create the FIFO special files specified by the operands, in the order
96728 specified.96729 For each *file* operand, the *mkfifo* utility shall perform actions equivalent to the *mkfifo()* function
96730 defined in the System Interfaces volume of POSIX.1-2008, called with the following arguments:

- 96731 1. The *file* operand is used as the *path* argument.
- 96732 2. The value of the bitwise-inclusive OR of S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP,
96733 S_IROTH, and S_IWOTH is used as the *mode* argument. (If the **-m** option is specified, the
96734 value of the *mkfifo()* *mode* argument is unspecified, but the FIFO shall at no time have
96735 permissions less restrictive than the **-m mode** option-argument.)

96736 **OPTIONS**96737 The *mkfifo* utility shall conform to XBD [Section 12.2](#) (on page 215).

96738 The following option shall be supported:

96739 **-m mode** Set the file permission bits of the newly-created FIFO to the specified *mode* value.
 96740 The *mode* option-argument shall be the same as the *mode* operand defined for the
 96741 *chmod* utility. In the *symbolic_mode* strings, the *op* characters '+' and '-' shall be
 96742 interpreted relative to an assumed initial mode of *a=rw*.

96743 **OPERANDS**

96744 The following operand shall be supported:

96745 *file* A pathname of the FIFO special file to be created.96746 **STDIN**

96747 Not used.

96748 **INPUT FILES**

96749 None.

96750 **ENVIRONMENT VARIABLES**96751 The following environment variables shall affect the execution of *mkfifo*:

96752 **LANG** Provide a default value for the internationalization variables that are unset or null.
 96753 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 96754 variables used to determine the values of locale categories.)

96755 **LC_ALL** If set to a non-empty string value, override the values of all the other
 96756 internationalization variables.

96757 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 96758 characters (for example, single-byte as opposed to multi-byte characters in
 96759 arguments).

96760 **LC_MESSAGES**

96761 Determine the locale that should be used to affect the format and contents of
 96762 diagnostic messages written to standard error.

Utilities

mkfifo

96763 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

96764 **ASYNCHRONOUS EVENTS**

96765 Default.

96766 **STDOUT**

96767 Not used.

96768 **STDERR**

96769 The standard error shall be used only for diagnostic messages.

96770 **OUTPUT FILES**

96771 None.

96772 **EXTENDED DESCRIPTION**

96773 None.

96774 **EXIT STATUS**

96775 The following exit values shall be returned:

96776 0 All the specified FIFO special files were created successfully.

96777 >0 An error occurred.

96778 **CONSEQUENCES OF ERRORS**

96779 Default.

96780 **APPLICATION USAGE**

96781 None.

96782 **EXAMPLES**

96783 None.

96784 **RATIONALE**

96785 This utility was added to permit shell applications to create FIFO special files.

96786 The **-m** option was added to control the file mode, for consistency with the similar functionality provided by the *mkdir* utility.

96788 Early proposals included a **-p** option similar to the *mkdir -p* option that created intermediate directories leading up to the FIFO specified by the final component. This was removed because it is not commonly needed and is not common practice with similar utilities.

96791 The functionality of *mkfifo* is described substantially through a reference to the *mkfifo()* function in the System Interfaces volume of POSIX.1-2008. For example, by default, the mode of the FIFO file is affected by the file mode creation mask in accordance with the specified behavior of the *mkfifo()* function. In this way, there is less duplication of effort required for describing details of the file creation.

96796 **FUTURE DIRECTIONS**

96797 None.

96798 **SEE ALSO**

96799 *chmod*, *umask*

96800 XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

96801 XSH *mkfifo()*

96802	CHANGE HISTORY
96803	First released in Issue 3.
96804	Issue 6
96805	The <code>-m</code> option is aligned with the IEEE P1003.2b draft standard to clarify an ambiguity.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

more

NAME

more — display files on a page-by-page basis

SYNOPSIS

UP `more [-ceisu] [-n number] [-p command] [-t tagstring] [file...]`

DESCRIPTION

The *more* utility shall read files and either write them to the terminal on a page-by-page basis or filter them to standard output. If standard output is not a terminal device, all input files shall be copied to standard output in their entirety, without modification, except as specified for the `-s` option. If standard output is a terminal device, the files shall be written a number of lines (one screenful) at a time under the control of user commands. See the EXTENDED DESCRIPTION section.

Certain block-mode terminals do not have all the capabilities necessary to support the complete *more* definition; they are incapable of accepting commands that are not terminated with a <newline>. Implementations that support such terminals shall provide an operating mode to *more* in which all commands can be terminated with a <newline> on those terminals. This mode:

- Shall be documented in the system documentation
- Shall, at invocation, inform the user of the terminal deficiency that requires the <newline> usage and provide instructions on how this warning can be suppressed in future invocations
- Shall not be required for implementations supporting only fully capable terminals
- Shall not affect commands already requiring <newline> characters
- Shall not affect users on the capable terminals from using *more* as described in this volume of POSIX.1-2008

OPTIONS

The *more* utility shall conform to XBD Section 12.2 (on page 215), except that '+' may be recognized as an option delimiter as well as '-'.

The following options shall be supported:

- c** If a screen is to be written that has no lines in common with the current screen, or *more* is writing its first screen, *more* shall not scroll the screen, but instead shall redraw each line of the screen in turn, from the top of the screen to the bottom. In addition, if *more* is writing its first screen, the screen shall be cleared. This option may be silently ignored on devices with insufficient terminal capabilities.
- e** By default, *more* shall exit immediately after writing the last line of the last file in the argument list. If the `-e` option is specified:
 1. If there is only a single file in the argument list and that file was completely displayed on a single screen, *more* shall exit immediately after writing the last line of that file.
 2. Otherwise, *more* shall exit only after reaching end-of-file on the last file in the argument list twice without an intervening operation. See the EXTENDED DESCRIPTION section.
- i** Perform pattern matching in searches without regard to case; see XBD Section 9.2 (on page 182).

more*Utilities*

- 96848 **-n number** Specify the number of lines per screenful. The *number* argument is a positive
 96849 decimal integer. The **-n** option shall override any values obtained from any other
 96850 source.
- 96851 **-p command** Each time a screen from a new file is displayed or redisplayed (including as a
 96852 result of *more* commands; for example, **:p**), execute the *more* command(s) in the
 96853 command arguments in the order specified, as if entered by the user after the first
 96854 screen has been displayed. No intermediate results shall be displayed (that is, if the
 96855 command is a movement to a screen different from the normal first screen, only the
 96856 screen resulting from the command shall be displayed.) If any of the commands
 96857 fail for any reason, an informational message to this effect shall be written, and no
 96858 further commands specified using the **-p** option shall be executed for this file.
- 96859 **-s** Behave as if consecutive empty lines were a single empty line.
- 96860 **-t tagstring** Write the screenful of the file containing the tag named by the *tagstring* argument.
 96861 See the *ctags* utility. The tags feature represented by **-t tagstring** and the **:t**
 96862 command is optional. It shall be provided on any system that also provides a
 96863 conforming implementation of *ctags*; otherwise, the use of **-t** produces undefined
 96864 results.
- 96865 The filename resulting from the **-t** option shall be logically added as a prefix to the
 96866 list of command line files, as if specified by the user. If the tag named by the
 96867 *tagstring* argument is not found, it shall be an error, and *more* shall take no further
 96868 action.
- 96869 If the tag specifies a line number, the first line of the display shall contain the
 96870 beginning of that line. If the tag specifies a pattern, the first line of the display shall
 96871 contain the beginning of the matching text from the first line of the file that
 96872 contains that pattern. If the line does not exist in the file or matching text is not
 96873 found, an informational message to this effect shall be displayed, and *more* shall
 96874 display the default screen as if **-t** had not been specified.
- 96875 If both the **-t tagstring** and **-p command** options are given, the **-t tagstring** shall be
 96876 processed first; that is, the file and starting line for the display shall be as specified
 96877 by **-t**, and then the **-p more** command shall be executed. If the line (matching text)
 96878 specified by the **-t** command does not exist (is not found), no **-p more** command
 96879 shall be executed for this file at any time.
- 96880 **-u** Treat a <backspace> as a printable control character, displayed as an
 96881 implementation-defined character sequence (see the EXTENDED DESCRIPTION
 96882 section), suppressing backspacing and the special handling that produces
 96883 underlined or standout mode text on some terminal types. Also, do not ignore a
 96884 <carriage-return> at the end of a line.

OPERANDS

The following operand shall be supported:

- 96887 *file* A pathname of an input file. If no *file* operands are specified, the standard input
 96888 shall be used. If a *file* is ' - ', the standard input shall be read at that point in the
 96889 sequence.

STDIN

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is ' - '.

INPUT FILES

The input files being examined shall be text files. If standard output is a terminal, standard error shall be used to read commands from the user. If standard output is a terminal, standard error is not readable, and command input is needed, *more* may attempt to obtain user commands from the controlling terminal (for example, */dev/tty*); otherwise, *more* shall terminate with an error indicating that it was unable to read user commands. If standard output is not a terminal, no error shall result if standard error cannot be opened for reading.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *more*:

COLUMNS Override the system-selected horizontal display line size. See XBD Chapter 8 (on page 173) for valid values and results when it is unset or null.

EDITOR Used by the *v* command to select an editor. See the EXTENDED DESCRIPTION section.

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE

Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

LINES Override the system-selected vertical screen size, used as the number of lines in a screenful. See XBD Chapter 8 (on page 173) for valid values and results when it is unset or null. The *-n* option shall take precedence over the *LINES* variable for determining the number of lines in a screenful.

MORE

Determine a string containing options described in the OPTIONS section preceded with <hyphen> characters and <blank>-separated as on the command line. Any command line options shall be processed after those in the *MORE* variable, as if the command line were:

more \$MORE options operands

The *MORE* variable shall take precedence over the *TERM* and *LINES* variables for determining the number of lines in a screenful.

TERM

Determine the name of the terminal type. If this variable is unset or null, an unspecified default terminal type is used.

more*Utilities*96935 **ASYNCHRONOUS EVENTS**

96936 Default.

96937 **STDOUT**

96938 The standard output shall be used to write the contents of the input files.

96939 **STDERR**

96940 The standard error shall be used for diagnostic messages and user commands (see the INPUT
 96941 FILES section), and, if standard output is a terminal device, to write a prompting string. The
 96942 prompting string shall appear on the screen line below the last line of the file displayed in the
 96943 current screenful. The prompt shall contain the name of the file currently being examined and
 96944 shall contain an end-of-file indication and the name of the next file, if any, when prompting at
 96945 the end-of-file. If an error or informational message is displayed, it is unspecified whether it is
 96946 contained in the prompt. If it is not contained in the prompt, it shall be displayed and then the
 96947 user shall be prompted for a continuation character, at which point another message or the user
 96948 prompt may be displayed. The prompt is otherwise unspecified. It is unspecified whether
 96949 informational messages are written for other user commands.

96950 **OUTPUT FILES**

96951 None.

96952 **EXTENDED DESCRIPTION**

96953 The following section describes the behavior of *more* when the standard output is a terminal
 96954 device. If the standard output is not a terminal device, no options other than **-s** shall have any
 96955 effect, and all input files shall be copied to standard output otherwise unmodified, at which time
 96956 *more* shall exit without further action.

96957 The number of lines available per screen shall be determined by the **-n** option, if present, or by
 96958 examining values in the environment (see the ENVIRONMENT VARIABLES section). If neither
 96959 method yields a number, an unspecified number of lines shall be used.

96960 The maximum number of lines written shall be one less than this number, because the screen
 96961 line after the last line written shall be used to write a user prompt and user input. If the number
 96962 of lines in the screen is less than two, the results are undefined. It is unspecified whether user
 96963 input is permitted to be longer than the remainder of the single line where the prompt has been
 96964 written.

96965 The number of columns available per line shall be determined by examining values in the
 96966 environment (see the ENVIRONMENT VARIABLES section), with a default value as described
 96967 in XBD Chapter 8 (on page 173).

96968 Lines that are longer than the display shall be folded; the length at which folding occurs is
 96969 unspecified, but should be appropriate for the output device. Folding may occur between glyphs
 96970 of single characters that take up multiple display columns.

96971 When standard output is a terminal and **-u** is not specified, *more* shall treat <backspace> and
 96972 <carriage-return> characters specially:

- 96973 • A character, followed first by a sequence of *n* <backspace> characters (where *n* is the same
 96974 as the number of column positions that the character occupies), then by *n* <underscore>
 96975 characters (' _ '), shall cause that character to be written as underlined text, if the terminal
 96976 type supports that. The *n* <underscore> characters, followed first by *n* <backspace>
 96977 characters, then any character with *n* column positions, shall also cause that character to be
 96978 written as underlined text, if the terminal type supports that.

- A sequence of n <backspace> characters (where n is the same as the number of column positions that the previous character occupies) that appears between two identical printable characters shall cause the first of those two characters to be written as emboldened text (that is, visually brighter, standout mode, or inverse-video mode), if the terminal type supports that, and the second to be discarded. Immediately subsequent occurrences of <backspace>/character pairs for that same character shall also be discarded. (For example, the sequence "a\ba\ba\ba" is interpreted as a single emboldened 'a'.)
- The *more* utility shall logically discard all other <backspace> characters from the line as well as the character which precedes them, if any.
- A <carriage-return> at the end of a line shall be ignored, rather than being written as a non-printable character, as described in the next paragraph.

It is implementation-defined how other non-printable characters are written. Implementations should use the same format that they use for the *ex print* command; see the OPTIONS section within the *ed* utility. It is unspecified whether a multi-column character shall be separated if it crosses a display line boundary; it shall not be discarded. The behavior is unspecified if the number of columns on the display is less than the number of columns any single character in the line being displayed would occupy.

When each new file is displayed (or redisplayed), *more* shall write the first screen of the file. Once the initial screen has been written, *more* shall prompt for a user command. If the execution of the user command results in a screen that has lines in common with the current screen, and the device has sufficient terminal capabilities, *more* shall scroll the screen; otherwise, it is unspecified whether the screen is scrolled or redrawn.

For all files but the last (including standard input if no file was specified, and for the last file as well, if the *-e* option was not specified), when *more* has written the last line in the file, *more* shall prompt for a user command. This prompt shall contain the name of the next file as well as an indication that *more* has reached end-of-file. If the user command is *f*, <control>-F, <space>, *j*, <newline>, *d*, <control>-D, or *s*, *more* shall display the next file. Otherwise, if displaying the last file, *more* shall exit. Otherwise, *more* shall execute the user command specified.

Several of the commands described in this section display a previous screen from the input stream. In the case that text is being taken from a non-rewindable stream, such as a pipe, it is implementation-defined how much backwards motion is supported. If a command cannot be executed because of a limitation on backwards motion, an error message to this effect shall be displayed, the current screen shall not change, and the user shall be prompted for another command.

If a command cannot be performed because there are insufficient lines to display, *more* shall alert the terminal. If a command cannot be performed because there are insufficient lines to display or a */* command fails: if the input is the standard input, the last screen in the file may be displayed; otherwise, the current file and screen shall not change, and the user shall be prompted for another command.

The interactive commands in the following sections shall be supported. Some commands can be preceded by a decimal integer, called *count* in the following descriptions. If not specified with the command, *count* shall default to 1. In the following descriptions, *pattern* is a basic regular expression, as described in XBD Section 9.3 (on page 183). The term "examine" is historical usage meaning "open the file for viewing"; for example, *more foo* would be expressed as examining file *foo*.

In the following descriptions, unless otherwise specified, *line* is a line in the *more* display, not a

97026 line from the file being examined.

97027 In the following descriptions, the *current position* refers to two things:

- 97028 1. The position of the current line on the screen
- 97029 2. The line number (in the file) of the current line on the screen

97030 Usually, the line on the screen corresponding to the current position is the third line on the
 97031 screen. If this is not possible (there are fewer than three lines to display or this is the first page of
 97032 the file, or it is the last page of the file), then the current position is either the first or last line on
 97033 the screen as described later.

97034 Help

97035 *Synopsis:* h

97036 Write a summary of these commands and other implementation-defined commands. The
 97037 behavior shall be as if the *more* utility were executed with the *-e* option on a file that contained
 97038 the summary information. The user shall be prompted as described earlier in this section when
 97039 end-of-file is reached. If the user command is one of those specified to continue to the next file,
 97040 *more* shall return to the file and screen state from which the *h* command was executed.

97041 Scroll Forward One Screenful

97042 *Synopsis:* [*count*] f
 97043 [*count*] <control>--F

97044 Scroll forward *count* lines, with a default of one screenful. If *count* is more than the screen size,
 97045 only the final screenful shall be written.

97046 Scroll Backward One Screenful

97047 *Synopsis:* [*count*] b
 97048 [*count*] <control>--B

97049 Scroll backward *count* lines, with a default of one screenful (see the *-n* option). If *count* is more
 97050 than the screen size, only the final screenful shall be written.

97051 Scroll Forward One Line

97052 *Synopsis:* [*count*] <space>
 97053 [*count*] j
 97054 [*count*] <newline>

97055 Scroll forward *count* lines. The default *count* for the <space> shall be one screenful; for *j* and
 97056 <newline>, one line. The entire *count* lines shall be written, even if *count* is more than the screen
 97057 size.

97058 Scroll Backward One Line

97059 *Synopsis:* [*count*] k

97060 Scroll backward *count* lines. The entire *count* lines shall be written, even if *count* is more than the
 97061 screen size.

Scroll Forward One Half Screenful

Synopsis: [*count*]d
 [*count*]<control>-D

Scroll forward *count* lines, with a default of one half of the screen size. If *count* is specified, it shall become the new default for subsequent **d**, <control>-D, and **u** commands.

Skip Forward One Line

Synopsis: [*count*]s

Display the screenful beginning with the line *count* lines after the last line on the current screen. If *count* would cause the current position to be such that less than one screenful would be written, the last screenful in the file shall be written.

Scroll Backward One Half Screenful

Synopsis: [*count*]u
 [*count*]<control>-U

Scroll backward *count* lines, with a default of one half of the screen size. If *count* is specified, it shall become the new default for subsequent **d**, <control>-D, **u**, and <control>-U commands. The entire *count* lines shall be written, even if *count* is more than the screen size.

Go to Beginning of File

Synopsis: [*count*]g

Display the screenful beginning with line *count*.

Go to End-of-File

Synopsis: [*count*]G

If *count* is specified, display the screenful beginning with the line *count*. Otherwise, display the last screenful of the file.

Refresh the Screen

Synopsis: r
 <control>-L

Refresh the screen.

Discard and Refresh

Synopsis: R

Refresh the screen, discarding any buffered input. If the current file is non-seekable, buffered input shall not be discarded and the **R** command shall be equivalent to the **r** command.

97093 **Mark Position**97094 *Synopsis:* `mletter`

97095 Mark the current position with the letter named by *letter*, where *letter* represents the name of one
 97096 of the lowercase letters of the portable character set. When a new file is examined, all marks may
 97097 be lost.

97098 **Return to Mark**97099 *Synopsis:* `' letter`

97100 Return to the position that was previously marked with the letter named by *letter*, making that
 97101 line the current position.

97102 **Return to Previous Position**97103 *Synopsis:* `' '`

97104 Return to the position from which the last large movement command was executed (where a
 97105 "large movement" is defined as any movement of more than a screenful of lines). If no such
 97106 movements have been made, return to the beginning of the file.

97107 **Search Forward for Pattern**97108 *Synopsis:* `[count]/[!]pattern<newline>`

97109 Display the screenful beginning with the *count*th line containing the pattern. The search shall
 97110 start after the first line currently displayed. The null regular expression (`' / '` followed by a
 97111 `<newline>`) shall repeat the search using the previous regular expression, with a default *count*. If
 97112 the character `' ! '` is included, the matching lines shall be those that do not contain the *pattern*. If
 97113 no match is found for the *pattern*, a message to that effect shall be displayed.

97114 **Search Backward for Pattern**97115 *Synopsis:* `[count]?[!]pattern<newline>`

97116 Display the screenful beginning with the *count*th previous line containing the pattern. The search
 97117 shall start on the last line before the first line currently displayed. The null regular expression
 97118 (`' ? '` followed by a `<newline>`) shall repeat the search using the previous regular expression,
 97119 with a default *count*. If the character `' ! '` is included, matching lines shall be those that do not
 97120 contain the *pattern*. If no match is found for the *pattern*, a message to that effect shall be
 97121 displayed.

97122 **Repeat Search**97123 *Synopsis:* `[count]n`

97124 Repeat the previous search for *count*th line containing the last *pattern* (or not containing the last
 97125 *pattern*, if the previous search was `" / "` or `" ? ! "`).

Repeat Search in Reverse

Synopsis: [*count*]*N*

Repeat the search in the opposite direction of the previous search for the *count*th line containing the last *pattern* (or not containing the last *pattern*, if the previous search was "/" or "?").

Examine New File

Synopsis: :e [*filename*]*<newline>*

Examine a new file. If the *filename* argument is not specified, the current file (see the *n* and *p* commands below) shall be re-examined. The *filename* shall be subjected to the process of shell word expansions (see Section 2.6, on page 2305); if more than a single pathname results, the effects are unspecified. If *filename* is a <number-sign> ('#'), the previously examined file shall be re-examined. If *filename* is not accessible for any reason (including that it is a non-seekable file), an error message to this effect shall be displayed and the current file and screen shall not change.

Examine Next File

Synopsis: [*count*]:*n*

Examine the next file. If a number *count* is specified, the *count*th next file shall be examined. If *filename* refers to a non-seekable file, the results are unspecified.

Examine Previous File

Synopsis: [*count*]:*p*

Examine the previous file. If a number *count* is specified, the *count*th previous file shall be examined. If *filename* refers to a non-seekable file, the results are unspecified.

Go to Tag

Synopsis: :t *tagstring**<newline>*

If the file containing the tag named by the *tagstring* argument is not the current file, examine the file, as if the *e* command was executed with that file as the argument. Otherwise, or in addition, display the screenful beginning with the tag, as described for the *-t* option (see the OPTIONS section). If the *ctags* utility is not supported by the system, the use of *:t* produces undefined results.

Invoke Editor

Synopsis: v

Invoke an editor to edit the current file being examined. If standard input is being examined, the results are unspecified. The name of the editor shall be taken from the environment variable *EDITOR*, or shall default to *vi*. If the last pathname component in *EDITOR* is either *vi* or *ex*, the editor shall be invoked with a *-c linenumber* command line argument, where *linenumber* is the line number of the file line containing the display line currently displayed as the first line of the screen. It is implementation-defined whether line-setting options are passed to editors other than *vi* and *ex*.

When the editor exits, *more* shall resume with the same file and screen as when the editor was invoked.

97165 **Display Position**

97166 *Synopsis:* =

97167 <control>--G

97168 Write a message for which the information references the first byte of the line after the last line of

97169 the file on the screen. This message shall include the name of the file currently being examined,

97170 its number relative to the total number of files there are to examine, the line number in the file,

97171 the byte number and the total bytes in the file, and what percentage of the file precedes the

97172 current position. If *more* is reading from standard input, or the file is shorter than a single screen,

97173 the line number, the byte number, the total bytes, and the percentage need not be written.

97174 **Quit**

97175 *Synopsis:* q

97176 :q

97177 ZZ

97178 Exit *more*.

97179 **EXIT STATUS**

97180 The following exit values shall be returned:

97181 0 Successful completion.

97182 >0 An error occurred.

97183 **CONSEQUENCES OF ERRORS**

97184 If an error is encountered accessing a file when using the **:n** command, *more* shall attempt to

97185 examine the next file in the argument list, but the final exit status shall be affected. If an error is

97186 encountered accessing a file via the **:p** command, *more* shall attempt to examine the previous file

97187 in the argument list, but the final exit status shall be affected. If an error is encountered accessing

97188 a file via the **:e** command, *more* shall remain in the current file and the final exit status shall not

97189 be affected.

97190 **APPLICATION USAGE**

97191 When the standard output is not a terminal, only the **-s** filter-modification option is effective.

97192 This is based on historical practice. For example, a typical implementation of *man* pipes its

97193 output through *more -s* to squeeze excess white space for terminal users. When *man* is piped to

97194 *lp*, however, it is undesirable for this squeezing to happen.

97195 **EXAMPLES**

97196 The **-p** allows arbitrary commands to be executed at the start of each file. Examples are:

97197 *more -p G file1 file2*

97198 Examine each file starting with its last screenful.

97199 *more -p 100 file1 file2*

97200 Examine each file starting with line 100 in the current position (usually the third line, so line

97201 98 would be the first line written).

97202 *more -p /100 file1 file2*

97203 Examine each file starting with the first line containing the string "100" in the current

97204 position

RATIONALE

The *more* utility, available in BSD and BSD-derived systems, was chosen as the prototype for the POSIX file display program since it is more widely available than either the public-domain program *less* or than *pg*, a pager provided in System V. The 4.4 BSD *more* is the model for the features selected; it is almost fully upwards-compatible from the 4.3 BSD version in wide use and has become more amenable for *vi* users. Several features originally derived from various file editors, found in both *less* and *pg*, have been added to this volume of POSIX.1-2008 as they have proved extremely popular with users.

There are inconsistencies between *more* and *vi* that result from historical practice. For example, the single-character commands **h**, **f**, **b**, and **<space>** are screen movers in *more*, but cursor movers in *vi*. These inconsistencies were maintained because the cursor movements are not applicable to *more* and the powerful functionality achieved without the use of the control key justifies the differences.

The tags interface has been included in a program that is not a text editor because it promotes another degree of consistent operation with *vi*. It is conceivable that the paging environment of *more* would be superior for browsing source code files in some circumstances.

The operating mode referred to for block-mode terminals effectively adds a **<newline>** to each Synopsis line that currently has none. So, for example, **d<newline>** would page one screenful. The mode could be triggered by a command line option, environment variable, or some other method. The details are not imposed by this volume of POSIX.1-2008 because there are so few systems known to support such terminals. Nevertheless, it was considered that all systems should be able to support *more* given the exception cited for this small community of terminals because, in comparison to *vi*, the cursor movements are few and the command set relatively amenable to the optional **<newline>** characters.

Some versions of *more* provide a shell escaping mechanism similar to the *ex !* command. The standard developers did not consider that this was necessary in a paginator, particularly given the wide acceptance of multiple window terminals and job control features. (They chose to retain such features in the editors and *mailx* because the shell interaction also gives an opportunity to modify the editing buffer, which is not applicable to *more*.)

The **-p** (position) option replaces the **+** command because of the Utility Syntax Guidelines. The **+command** option is no longer specified by POSIX.1-2008 but may be present in some implementations. In early proposals, it took a *pattern* argument, but historical *less* provided the *more* general facility of a command. It would have been desirable to use the same **-c** as *ex* and *vi*, but the letter was already in use.

The text stating “from a non-rewindable stream ... implementations may limit the amount of backwards motion supported” would allow an implementation that permitted no backwards motion beyond text already on the screen. It was not possible to require a minimum amount of backwards motion that would be effective for all conceivable device types. The implementation should allow the user to back up as far as possible, within device and reasonable memory allocation constraints.

Historically, non-printable characters were displayed using the ARPA standard mappings, which are as follows:

1. Printable characters are left alone.
2. Control characters less than **\177** are represented as followed by the character offset from the **'@'** character in the ASCII map; for example, **\007** is represented as **'G'**.

- 97250 3. \177 is represented as followed by ' ? ' .
- 97251 The display of characters having their eighth bit set was less standard. Existing implementations
- 97252 use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed characters with their
- 97253 eighth bit set as the two characters "M-", followed by the seven-bit display as described
- 97254 previously.) The latter probably has the best claim to historical practice because it was used with
- 97255 the -v option of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.
- 97256 No specific display format is required by POSIX.1-2008. Implementations are encouraged to
- 97257 conform to historic practice in the absence of any strong reason to diverge.
- 97258 **FUTURE DIRECTIONS**
- 97259 None.
- 97260 **SEE ALSO**
- 97261 [Chapter 2](#) (on page 2297), *ctags*, *ed*, *ex*, *vi*
- 97262 XBD [Chapter 8](#) (on page 173), [Section 9.2](#) (on page 182), [Section 9.3](#) (on page 183), [Section 12.2](#)
- 97263 (on page 215)
- 97264 **CHANGE HISTORY**
- 97265 First released in Issue 4.
- 97266 **Issue 5**
- 97267 The FUTURE DIRECTIONS section is added.
- 97268 **Issue 6**
- 97269 This utility is marked as part of the User Portability Utilities option.
- 97270 The obsolescent SYNOPSIS is removed.
- 97271 The utility has been extensively reworked for alignment with the IEEE P1003.2b draft standard:
- 97272 • Changes have been made as a result of IEEE PASC Interpretations 1003.2 #37 and #109.
- 97273 • The *more* utility should be able to handle underlined and emboldened displays of
- 97274 characters that are wider than a single column position.
- 97275 **Issue 7**
- 97276 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized
- 97277 as an option delimiter in the OPTIONS section.
- 97278 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Utilities

mv

NAME

mv — move files

SYNOPSIS

mv [-if] *source_file target_file*

mv [-if] *source_file... target_dir*

DESCRIPTION

In the first synopsis form, the *mv* utility shall move the file named by the *source_file* operand to the destination specified by the *target_file*. This first synopsis form is assumed when the final operand does not name an existing directory and is not a symbolic link referring to an existing directory. In this case, if *target_file* ends with a trailing <slash> character, *mv* shall treat this as an error and no *source_file* operands will be processed.

In the second synopsis form, *mv* shall move each file named by a *source_file* operand to a destination file in the existing directory named by the *target_dir* operand, or referenced if *target_dir* is a symbolic link referring to an existing directory. The destination path for each *source_file* shall be the concatenation of the target directory, a single <slash> character if the target did not end in a <slash>, and the last pathname component of the *source_file*. This second form is assumed when the final operand names an existing directory.

If any operand specifies an existing file of a type not specified by the System Interfaces volume of POSIX.1-2008, the behavior is implementation-defined.

For each *source_file* the following steps shall be taken:

1. If the destination path exists, the *-f* option is not specified, and either of the following conditions is true:
 - a. The permissions of the destination path do not permit writing and the standard input is a terminal.
 - b. The *-i* option is specified.

the *mv* utility shall write a prompt to standard error and read a line from standard input. If the response is not affirmative, *mv* shall do nothing more with the current *source_file* and go on to any remaining *source_files*.

2. If the *source_file* operand and destination path name the same existing file, then the destination path shall not be removed, and one of the following shall occur:
 - a. No change is made to *source_file*, no error occurs, and no diagnostic is issued.
 - b. No change is made to *source_file*, a diagnostic is issued to standard error identifying the two names, and the exit status is affected.
 - c. If the *source_file* operand and destination path name distinct directory entries, then the *source_file* operand is removed, no error occurs, and no diagnostic is issued.

The *mv* utility shall do nothing more with the current *source_file*, and go on to any remaining *source_files*.

3. The *mv* utility shall perform actions equivalent to the *rename()* function defined in the System Interfaces volume of POSIX.1-2008, called with the following arguments:
 - a. The *source_file* operand is used as the *old* argument.
 - b. The destination path is used as the *new* argument.

If this succeeds, *mv* shall do nothing more with the current *source_file* and go on to any remaining *source_files*. If this fails for any reasons other than those described for the *errno*

- 97322 [EXDEV] in the System Interfaces volume of POSIX.1-2008, *mv* shall write a diagnostic
 97323 message to standard error, do nothing more with the current *source_file*, and go on to any
 97324 remaining *source_files*.
- 97325 4. If the destination path exists, and it is a file of type directory and *source_file* is not a file of
 97326 type directory, or it is a file not of type directory and *source_file* is a file of type directory,
 97327 *mv* shall write a diagnostic message to standard error, do nothing more with the current
 97328 *source_file*, and go on to any remaining *source_files*. If the destination path exists and was
 97329 created by a previous step, it is unspecified whether this will be treated as an error or the
 97330 destination path will be overwritten.
- 97331 5. If the destination path exists, *mv* shall attempt to remove it. If this fails for any reason, *mv*
 97332 shall write a diagnostic message to standard error, do nothing more with the current
 97333 *source_file*, and go on to any remaining *source_files*.
- 97334 6. The file hierarchy rooted in *source_file* shall be duplicated as a file hierarchy rooted in the
 97335 destination path. If *source_file* or any of the files below it in the hierarchy are symbolic
 97336 links, the links themselves shall be duplicated, including their contents, rather than any
 97337 files to which they refer. The following characteristics of each file in the file hierarchy
 97338 shall be duplicated:
- 97339 • The time of last data modification and time of last access
 - 97340 • The user ID and group ID
 - 97341 • The file mode
- 97342 If the user ID, group ID, or file mode of a regular file cannot be duplicated, the file mode
 97343 bits *S_ISUID* and *S_ISGID* shall not be duplicated.
- 97344 When files are duplicated to another file system, the implementation may require that the
 97345 process invoking *mv* has read access to each file being duplicated.
- 97346 If files being duplicated to another file system have hard links to other files, it is
 97347 unspecified whether the files copied to the new file system have the hard links preserved
 97348 or separate copies are created for the linked files.
- 97349 If the duplication of the file hierarchy fails for any reason, *mv* shall write a diagnostic
 97350 message to standard error, do nothing more with the current *source_file*, and go on to any
 97351 remaining *source_files*.
- 97352 If the duplication of the file characteristics fails for any reason, *mv* shall write a diagnostic
 97353 message to standard error, but this failure shall not cause *mv* to modify its exit status.
- 97354 7. The file hierarchy rooted in *source_file* shall be removed. If this fails for any reason, *mv*
 97355 shall write a diagnostic message to the standard error, do nothing more with the current
 97356 *source_file*, and go on to any remaining *source_files*.

97357 OPTIONS

- 97358 The *mv* utility shall conform to XBD [Section 12.2](#) (on page 215).
- 97359 The following options shall be supported:
- 97360 **-f** Do not prompt for confirmation if the destination path exists. Any previous
 97361 occurrence of the **-i** option is ignored.
- 97362 **-i** Prompt for confirmation if the destination path exists. Any previous occurrence of
 97363 the **-f** option is ignored.
- 97364 Specifying more than one of the **-f** or **-i** options shall not be considered an error. The last option

Utilities

mv

97365 specified shall determine the behavior of *mv*.

97366 **OPERANDS**

97367 The following operands shall be supported:

97368 *source_file* A pathname of a file or directory to be moved.

97369 *target_file* A new pathname for the file or directory being moved.

97370 *target_dir* A pathname of an existing directory into which to move the input files.

97371 **STDIN**

97372 The standard input shall be used to read an input line in response to each prompt specified in
97373 the STDERR section. Otherwise, the standard input shall not be used.

97374 **INPUT FILES**

97375 The input files specified by each *source_file* operand can be of any file type.

97376 **ENVIRONMENT VARIABLES**

97377 The following environment variables shall affect the execution of *mv*:

97378 *LANG* Provide a default value for the internationalization variables that are unset or null.
97379 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
97380 variables used to determine the values of locale categories.)

97381 *LC_ALL* If set to a non-empty string value, override the values of all the other
97382 internationalization variables.

97383 *LC_COLLATE*

97384 Determine the locale for the behavior of ranges, equivalence classes, and multi-
97385 character collating elements used in the extended regular expression defined for
97386 the **yesexpr** locale keyword in the *LC_MESSAGES* category.

97387 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
97388 characters (for example, single-byte as opposed to multi-byte characters in
97389 arguments and input files), the behavior of character classes used in the extended
97390 regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES*
97391 category.

97392 *LC_MESSAGES*

97393 Determine the locale used to process affirmative responses, and the locale used to
97394 affect the format and contents of diagnostic messages and prompts written to
97395 standard error.

97396 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

97397 **ASYNCHRONOUS EVENTS**

97398 Default.

97399 **STDOUT**

97400 Not used.

97401 **STDERR**

97402 Prompts shall be written to the standard error under the conditions specified in the
97403 DESCRIPTION section. The prompts shall contain the destination pathname, but their format is
97404 otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic
97405 messages.

OUTPUT FILES

The output files may be of any file type.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All input files were moved successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

If the copying or removal of *source_file* is prematurely terminated by a signal or error, *mv* may leave a partial copy of *source_file* at the source or destination. The *mv* utility shall not modify both *source_file* and the destination path simultaneously; termination at any point shall leave either *source_file* or the destination path complete.

APPLICATION USAGE

Some implementations mark for update the last file status change timestamp of renamed files and some do not. Applications which make use of the last file status change timestamp may behave differently with respect to renamed files unless they are designed to allow for either behavior.

The specification ensures that *mv a a* will not alter the contents of file *a*, and allows the implementation to issue an error that a file cannot be moved onto itself. Likewise, when *a* and *b* are hard links to the same file, *mv a b* will not alter *b*, but if a diagnostic is not issued, then it is unspecified whether *a* is left untouched (as it would be by the *rename()* function) or unlinked (reducing the link count of *b*).

EXAMPLES

If the current directory contains only files *a* (of any type defined by the System Interfaces volume of POSIX.1-2008), *b* (also of any type), and a directory *c*:

```
mv a b c
```

```
mv c d
```

results with the original files *a* and *b* residing in the directory *d* in the current directory.

RATIONALE

Early proposals diverged from the SVID and BSD historical practice in that they required that when the destination path exists, the *-f* option is not specified, and input is not a terminal, *mv* fails. This was done for compatibility with *cp*. The current text returns to historical practice. It should be noted that this is consistent with the *rename()* function defined in the System Interfaces volume of POSIX.1-2008, which does not require write permission on the target.

For absolute clarity, paragraph (1), describing the behavior of *mv* when prompting for confirmation, should be interpreted in the following manner:

```
if (exists AND (NOT f_option) AND
    ((not_writable AND input_is_terminal) OR i_option))
```

The *-i* option exists on BSD systems, giving applications and users a way to avoid accidentally unlinking files when moving others. When the standard input is not a terminal, the 4.3 BSD *mv* deletes all existing destination paths without prompting, even when *-i* is specified; this is inconsistent with the behavior of the 4.3 BSD *cp* utility, which always generates an error when the file is unwritable and the standard input is not a terminal. The standard developers decided that use of *-i* is a request for interaction, so when the destination path exists, the utility takes

instructions from whatever responds to standard input.

The *rename()* function is able to move directories within the same file system. Some historical versions of *mv* have been able to move directories, but not to a different file system. The standard developers considered that this was an annoying inconsistency, so this volume of POSIX.1-2008 requires directories to be able to be moved even across file systems. There is no **-R** option to confirm that moving a directory is actually intended, since such an option was not required for moving directories in historical practice. Requiring the application to specify it sometimes, depending on the destination, seemed just as inconsistent. The semantics of the *rename()* function were preserved as much as possible. For example, *mv* is not permitted to “rename” files to or from directories, even though they might be empty and removable.

Historic implementations of *mv* did not exit with a non-zero exit status if they were unable to duplicate any file characteristics when moving a file across file systems, nor did they write a diagnostic message for the user. The former behavior has been preserved to prevent scripts from breaking; a diagnostic message is now required, however, so that users are alerted that the file characteristics have changed.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because implementations may desire more descriptive prompts than those used on historical implementations. Therefore, an application not using the **-f** option or using the **-i** option relies on the system to provide the most suitable dialog directly with the user, based on the behavior specified.

When *mv* is dealing with a single file system and *source_file* is a symbolic link, the link itself is moved as a consequence of the dependence on the *rename()* functionality, per the DESCRIPTION. Across file systems, this has to be made explicit.

FUTURE DIRECTIONS

None.

SEE ALSO

cp, *ln*

XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

XSH *rename()*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The *mv* utility is changed to describe processing of symbolic links as specified in the IEEE P1003.2b draft standard.

The APPLICATION USAGE section is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #016 is applied.

Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the *LC_MESSAGES* environment variable.

Austin Group Interpretations 1003.1-2001 #164, #168, and #169 are applied.

SD5-XCU-ERN-13 is applied, making an editorial correction to the SYNOPSIS.

SD5-XCU-ERN-51 is applied to the DESCRIPTION, defining the behavior for when files are being duplicated to another file system while having hard links.

97494

Changes are made related to support for finegrained timestamps.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

Utilities

newgrp**NAME**

newgrp — change to a new group

SYNOPSIS

newgrp [-l] [*group*]

DESCRIPTION

The *newgrp* utility shall create a new shell execution environment with a new real and effective group identification. Of the attributes listed in [Section 2.12](#) (on page 2331), the new shell execution environment shall retain the working directory, file creation mask, and exported variables from the previous environment (that is, open files, traps, unexported variables, alias definitions, shell functions, and *set* options may be lost). All other aspects of the process environment that are preserved by the *exec* family of functions defined in the System Interfaces volume of POSIX.1-2008 shall also be preserved by *newgrp*; whether other aspects are preserved is unspecified.

A failure to assign the new group identifications (for example, for security or password-related reasons) shall not prevent the new shell execution environment from being created.

The *newgrp* utility shall affect the supplemental groups for the process as follows:

- On systems where the effective group ID is normally in the supplementary group list (or whenever the old effective group ID actually is in the supplementary group list):
 - If the new effective group ID is also in the supplementary group list, *newgrp* shall change the effective group ID.
 - If the new effective group ID is not in the supplementary group list, *newgrp* shall add the new effective group ID to the list, if there is room to add it.
- On systems where the effective group ID is not normally in the supplementary group list (or whenever the old effective group ID is not in the supplementary group list):
 - If the new effective group ID is in the supplementary group list, *newgrp* shall delete it.
 - If the old effective group ID is not in the supplementary list, *newgrp* shall add it if there is room.

Note: The System Interfaces volume of POSIX.1-2008 does not specify whether the effective group ID of a process is included in its supplementary group list.

With no operands, *newgrp* shall change the effective group back to the groups identified in the user's user entry, and shall set the list of supplementary groups to that set in the user's group database entries.

If the first argument is '-', the results are unspecified.

If a password is required for the specified group, and the user is not listed as a member of that group in the group database, the user shall be prompted to enter the correct password for that group. If the user is listed as a member of that group, no password shall be requested. If no password is required for the specified group, it is implementation-defined whether users not listed as members of that group can change to that group. Whether or not a password is required, implementation-defined system accounting or security mechanisms may impose additional authorization restrictions that may cause *newgrp* to write a diagnostic message and suppress the changing of the group identification.

newgrp

Utilities

97537 **OPTIONS**

97538 The *newgrp* utility shall conform to XBD [Section 12.2](#) (on page 215), except for the unspecified
 97539 usage of ' - '.

97540 The following option shall be supported:

97541 **-l** (The letter ell.) Change the environment to what would be expected if the user
 97542 actually logged in again.

97543 **OPERANDS**

97544 The following operand shall be supported:

97545 *group* A group name from the group database or a non-negative numeric group ID.
 97546 Specifies the group ID to which the real and effective group IDs shall be set. If
 97547 *group* is a non-negative numeric string and exists in the group database as a group
 97548 name (see *getgrnam()*), the numeric group ID associated with that group name
 97549 shall be used as the group ID.

97550 **STDIN**

97551 Not used.

97552 **INPUT FILES**

97553 The file */dev/tty* shall be used to read a single line of text for password checking, when one is
 97554 required.

97555 **ENVIRONMENT VARIABLES**

97556 The following environment variables shall affect the execution of *newgrp*:

97557 *LANG* Provide a default value for the internationalization variables that are unset or null.
 97558 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 97559 variables used to determine the values of locale categories.)

97560 *LC_ALL* If set to a non-empty string value, override the values of all the other
 97561 internationalization variables.

97562 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 97563 characters (for example, single-byte as opposed to multi-byte characters in
 97564 arguments).

97565 *LC_MESSAGES*
 97566 Determine the locale that should be used to affect the format and contents of
 97567 diagnostic messages written to standard error.

97568 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

97569 **ASYNCHRONOUS EVENTS**

97570 Default.

97571 **STDOUT**

97572 Not used.

97573 **STDERR**

97574 The standard error shall be used for diagnostic messages and a prompt string for a password, if
 97575 one is required. Diagnostic messages may be written in cases where the exit status is not
 97576 available. See the EXIT STATUS section.

97577 **OUTPUT FILES**

97578 None.

97579 **EXTENDED DESCRIPTION**

97580 None.

97581 **EXIT STATUS**

97582 If *newgrp* succeeds in creating a new shell execution environment, whether or not the group
 97583 identification was changed successfully, the exit status shall be the exit status of the shell.
 97584 Otherwise, the following exit value shall be returned:

97585 >0 An error occurred.

97586 **CONSEQUENCES OF ERRORS**

97587 The invoking shell may terminate.

97588 **APPLICATION USAGE**

97589 There is no convenient way to enter a password into the group database. Use of group
 97590 passwords is not encouraged, because by their very nature they encourage poor security
 97591 practices. Group passwords may disappear in the future.

97592 A common implementation of *newgrp* is that the current shell uses *exec* to overlay itself with
 97593 *newgrp*, which in turn overlays itself with a new shell after changing group. On some
 97594 implementations, however, this may not occur and *newgrp* may be invoked as a subprocess.

97595 The *newgrp* command is intended only for use from an interactive terminal. It does not offer a
 97596 useful interface for the support of applications.

97597 The exit status of *newgrp* is generally inapplicable. If *newgrp* is used in a script, in most cases it
 97598 successfully invokes a new shell and the rest of the original shell script is bypassed when the
 97599 new shell exits. Used interactively, *newgrp* displays diagnostic messages to indicate problems.
 97600 But usage such as:

97601 *newgrp* foo
 97602 *echo* \$?

97603 is not useful because the new shell might not have access to any status *newgrp* may have
 97604 generated (and most historical systems do not provide this status). A zero status echoed here
 97605 does not necessarily indicate that the user has changed to the new group successfully. Following
 97606 *newgrp* with the *id* command provides a portable means of determining whether the group
 97607 change was successful or not.

97608 **EXAMPLES**

97609 None.

97610 **RATIONALE**

97611 Most historical implementations use one of the *exec* functions to implement the behavior of
 97612 *newgrp*. Errors detected before the *exec* leave the environment unchanged, while errors detected
 97613 after the *exec* leave the user in a changed environment. While it would be useful to have *newgrp*
 97614 issue a diagnostic message to tell the user that the environment changed, it would be
 97615 inappropriate to require this change to some historical implementations.

97616 The password mechanism is allowed in the group database, but how this would be
 97617 implemented is not specified.

97618 The *newgrp* utility was retained in this volume of POSIX.1-2008, even given the existence of the
 97619 multiple group permissions feature in the System Interfaces volume of POSIX.1-2008, for several
 97620 reasons. First, in some implementations, the group ownership of a newly created file is
 97621 determined by the group of the directory in which the file is created, as allowed by the System

newgrp

Utilities

97622 Interfaces volume of POSIX.1-2008; on other implementations, the group ownership of a newly
 97623 created file is determined by the effective group ID. On implementations of the latter type,
 97624 *newgrp* allows files to be created with a specific group ownership. Finally, many
 97625 implementations use the real group ID in accounting, and on such systems, *newgrp* allows the
 97626 accounting identity of the user to be changed.

97627 **FUTURE DIRECTIONS**

97628 None.

97629 **SEE ALSO**97630 [Chapter 2](#) (on page 2297), *sh*97631 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)97632 XSH *exec*, *getgrnam()*97633 **CHANGE HISTORY**

97634 First released in Issue 2.

97635 **Issue 6**

97636 This utility is marked as part of the User Portability Utilities option.

97637 The obsolescent SYNOPSIS is removed.

97638 The text describing supplemental groups is no longer conditional on {NGROUPS_MAX} being
 97639 greater than 1. This is because {NGROUPS_MAX} now has a minimum value of 8. This is a FIPS
 97640 requirement.

97641 **Issue 7**

97642 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first
 97643 argument is ' - '.

97644 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

97645 The *newgrp* utility is moved from the User Portability Utilities option to the Base. User
 97646 Portability Utilities is now an option for interactive utilities.

Utilities

nice

NAME

nice — invoke a utility with an altered nice value

SYNOPSIS

`nice [-n increment] utility [argument...]`

DESCRIPTION

The *nice* utility shall invoke a utility, requesting that it be run with a different nice value (see XBD Section 3.239, on page 71). With no options, the executed utility shall be run with a nice value that is some implementation-defined quantity greater than or equal to the nice value of the current process. If the user lacks appropriate privileges to affect the nice value in the requested manner, the *nice* utility shall not affect the nice value; in this case, a warning message may be written to standard error, but this shall not prevent the invocation of *utility* or affect the exit status.

OPTIONS

The *nice* utility shall conform to XBD Section 12.2 (on page 215).

The following option is supported:

-n *increment* A positive or negative decimal integer which shall have the same effect on the execution of the utility as if the utility had called the *nice()* function with the numeric value of the *increment* option-argument.

OPERANDS

The following operands shall be supported:

utility The name of a utility that is to be invoked. If the *utility* operand names any of the special built-in utilities in Section 2.14 (on page 2334), the results are undefined.

argument Any string to be supplied as an argument when invoking the utility named by the *utility* operand.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *nice*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

nice*Utilities*

97689	<i>PATH</i>	Determine the search path used to locate the utility to be invoked. See XBD
97690		Chapter 8 (on page 173).
97691	ASYNCHRONOUS EVENTS	
97692		Default.
97693	STDOUT	
97694		Not used.
97695	STDERR	
97696		The standard error shall be used only for diagnostic messages.
97697	OUTPUT FILES	
97698		None.
97699	EXTENDED DESCRIPTION	
97700		None.
97701	EXIT STATUS	
97702		If <i>utility</i> is invoked, the exit status of <i>nice</i> shall be the exit status of <i>utility</i> ; otherwise, the <i>nice</i>
97703		utility shall exit with one of the following values:
97704	1-125	An error occurred in the <i>nice</i> utility.
97705	126	The utility specified by <i>utility</i> was found but could not be invoked.
97706	127	The utility specified by <i>utility</i> could not be found.
97707	CONSEQUENCES OF ERRORS	
97708		Default.
97709	APPLICATION USAGE	
97710		The only guaranteed portable uses of this utility are:
97711	<i>nice utility</i>	
97712		Run <i>utility</i> with the default higher or equal nice value.
97713	<i>nice -n <positive integer> utility</i>	
97714		Run <i>utility</i> with a higher nice value.
97715		On some implementations they have no discernible effect on the invoked utility and on some
97716		others they are exactly equivalent.
97717		Historical systems have frequently supported the <i><positive integer></i> up to 20. Since there is no
97718		error penalty associated with guessing a number that is too high, users without access to the
97719		system conformance document (to see what limits are actually in place) could use the historical 1
97720		to 20 range or attempt to use very large numbers if the job should be truly low priority.
97721		The nice value of a process can be displayed using the command:
97722	<i>ps -o nice</i>	
97723		The <i>command</i> , <i>env</i> , <i>nice</i> , <i>nohup</i> , <i>time</i> , and <i>xargs</i> utilities have been specified to use exit code 127 if
97724		an error occurs so that applications can distinguish “failure to find a utility” from “invoked
97725		utility exited with an error indication”. The value 127 was chosen because it is not commonly
97726		used for other meanings; most utilities use small values for “normal error conditions” and the
97727		values above 128 can be confused with termination due to receipt of a signal. The value 126 was
97728		chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
97729		scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
97730		between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
97731		<i>exec</i> the utility fail with [ENOENT], and uses 126 when any attempt to <i>exec</i> the utility fails for

97732 any other reason.

97733 EXAMPLES

97734 None.

97735 RATIONALE

97736 The 4.3 BSD version of *nice* does not check whether *increment* is a valid decimal integer. The
 97737 command *nice -x utility*, for example, would be treated the same as the command *nice --1*
 97738 *utility*. If the user does not have appropriate privileges, this results in a “permission denied”
 97739 error. This is considered a bug.

97740 When a user without appropriate privileges gives a negative *increment*, System V treats it like
 97741 the command *nice -0 utility*, while 4.3 BSD writes a “permission denied” message and does not
 97742 run the utility. The standard specifies the System V behavior together with an optional BSD-style
 97743 “permission denied” message.

97744 The C shell has a built-in version of *nice* that has a different interface from the one described in
 97745 this volume of POSIX.1-2008.

97746 The term “utility” is used, rather than “command”, to highlight the fact that shell compound
 97747 commands, pipelines, and so on, cannot be used. Special built-ins also cannot be used.
 97748 However, “utility” includes user application programs and shell scripts, not just utilities defined
 97749 in this volume of POSIX.1-2008.

97750 Historical implementations of *nice* provide a nice value range of 40 or 41 discrete steps, with the
 97751 default nice value being the midpoint of that range. By default, they raise the nice value of the
 97752 executed utility by 10.

97753 Some historical documentation states that the *increment* value must be within a fixed range. This
 97754 is misleading; the valid *increment* values on any invocation are determined by the current
 97755 process nice value, which is not always the default.

97756 The definition of nice value is not intended to suggest that all processes in a system have
 97757 priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the
 97758 System Interfaces volume of POSIX.1-2008 make the notion of a single underlying priority for all
 97759 scheduling policies problematic. Some implementations may implement the *nice*-related features
 97760 to affect all processes on the system, others to affect just the general time-sharing activities
 97761 implied by this volume of POSIX.1-2008, and others may have no effect at all. Because of the use
 97762 of “implementation-defined” in *nice* and *renice*, a wide range of implementation strategies are
 97763 possible.

97764 Earlier versions of this standard allowed a *-increment* option. This form is no longer specified by
 97765 POSIX.1-2008 but may be present in some implementations.

97766 FUTURE DIRECTIONS

97767 None.

97768 SEE ALSO

97769 [Chapter 2](#) (on page 2297), *renice*

97770 XBD [Section 3.239](#) (on page 71), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

97771 XSH *nice()*

97772 CHANGE HISTORY

97773 First released in Issue 4.

nice*Utilities*97774 **Issue 6**

97775 This utility is marked as part of the User Portability Utilities option.

97776 The obsolescent SYNOPSIS is removed.

97777 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/18 is applied, deleting a paragraph of
97778 RATIONALE that referred to text no longer in the standard.

97779 **Issue 7**

97780 Austin Group Interpretation 1003.1-2001 #027 is applied.

97781 SD5-XCU-ERN-32 and SD5-XCU-ERN-33 are applied, updating the DESCRIPTION,
97782 APPLICATION USAGE, and RATIONALE sections.

97783 The *nice* utility is moved from the User Portability Utilities option to the Base User Portability
97784 Utilities is now an option for interactive utilities.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

NAME

nl — line numbering filter

SYNOPSIS

```
nl [-p] [-b type] [-d delim] [-f type] [-h type] [-i incr] [-l num]
    [-n format] [-s sep] [-v startnum] [-w width] [file]
```

DESCRIPTION

The *nl* utility shall read lines from the named *file* or the standard input if no *file* is named and shall reproduce the lines to standard output. Lines shall be numbered on the left. Additional functionality may be provided in accordance with the command options in effect.

The *nl* utility views the text it reads in terms of logical pages. Line numbering shall be reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (for example, no numbering of header and footer lines while numbering blank lines only in the body).

The starts of logical page sections shall be signaled by input lines containing nothing but the following delimiter characters:

Line	Start of
\: \: \:	Header
\: \:	Body
\:	Footer

Unless otherwise specified, *nl* shall assume the text being read is in a single logical page body.

OPTIONS

The *nl* utility shall conform to XBD [Section 12.2](#) (on page 215). Only one file can be named.

The following options shall be supported:

-b type Specify which logical page body lines shall be numbered. Recognized *types* and their meaning are:

a Number all lines.

t Number only non-empty lines.

n No line numbering.

pstring Number only lines that contain the basic regular expression specified in *string*.

The default *type* for logical page body shall be **t** (text lines numbered).

-d delim Specify the delimiter characters that indicate the start of a logical page section. These can be changed from the default characters "\: " to two user-specified characters. If only one character is entered, the second character shall remain the default character ' : '.

-f type Specify the same as **b type** except for footer. The default for logical page footer shall be **n** (no lines numbered).

-h type Specify the same as **b type** except for header. The default *type* for logical page header shall be **n** (no lines numbered).

97825	-i <i>incr</i>	Specify the increment value used to number logical page lines. The default shall be 1.
97826		
97827	-l <i>num</i>	Specify the number of blank lines to be considered as one. For example, -l 2 results in only the second adjacent blank line being numbered (if the appropriate -h a , -b a , or -f a option is set). The default shall be 1.
97828		
97829		
97830	-n <i>format</i>	Specify the line numbering format. Recognized values are: ln , left justified, leading zeros suppressed; rn , right justified, leading zeros suppressed; rz , right justified, leading zeros kept. The default <i>format</i> shall be rn (right justified).
97831		
97832		
97833	-p	Specify that numbering should not be restarted at logical page delimiters.
97834	-s <i>sep</i>	Specify the characters used in separating the line number and the corresponding text line. The default <i>sep</i> shall be a <tab>.
97835		
97836	-v <i>startnum</i>	Specify the initial value used to number logical page lines. The default shall be 1.
97837	-w <i>width</i>	Specify the number of characters to be used for the line number. The default <i>width</i> shall be 6.
97838		

97839 OPERANDS

97840 The following operand shall be supported:

97841	<i>file</i>	A pathname of a text file to be line-numbered.

97842 STDIN

97843 The standard input shall be used if no *file* operand is specified, and shall be used if the *file*
 97844 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
 97845 the standard input shall not be used. See the INPUT FILES section.

97846 INPUT FILES

97847 The input file shall be a text file.

97848 ENVIRONMENT VARIABLES

97849 The following environment variables shall affect the execution of *nl*:

97850	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
97851		
97852		
97853	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
97854		
97855	<i>LC_COLLATE</i>	
97856		Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.
97857		
97858	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, and for deciding which characters are in character class graph (for the -b t , -f t , and -h t options).
97859		
97860		
97861		
97862		
97863	<i>LC_MESSAGES</i>	
97864		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
97865		

Utilities

nl

97866 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

97867 ASYNCHRONOUS EVENTS

97868 Default.

97869 STDOUT

97870 The standard output shall be a text file in the following format:

97871 "%s%s%s", <line number>, <separator>, <input line>

97872 where <line number> is one of the following numeric formats:

97873 %6d When the **rn** format is used (the default; see **-n**).

97874 %06d When the **rz** format is used.

97875 %-6d When the **ln** format is used.

97876 <empty> When line numbers are suppressed for a portion of the page, the <separator> is also suppressed.

97878 In the preceding list, the number 6 is the default width; the **-w** option can change this value.

97879 STDERR

97880 The standard error shall be used only for diagnostic messages.

97881 OUTPUT FILES

97882 None.

97883 EXTENDED DESCRIPTION

97884 None.

97885 EXIT STATUS

97886 The following exit values shall be returned:

97887 0 Successful completion.

97888 >0 An error occurred.

97889 CONSEQUENCES OF ERRORS

97890 Default.

97891 APPLICATION USAGE

97892 In using the **-d delim** option, care should be taken to escape characters that have special meaning to the command interpreter.

97894 EXAMPLES

97895 The command:

97896 `nl -v 10 -i 10 -d \!+ file1`

97897 numbers *file1* starting at line number 10 with an increment of 10. The logical page delimiter is "!.+". Note that the '!' has to be escaped when using *cs**h* as a command interpreter because of its history substitution syntax. For *ks**h* and *sh* the escape is not necessary, but does not do any harm.

97901 RATIONALE

97902 None.

97903 **FUTURE DIRECTIONS**

97904 None.

97905 **SEE ALSO**97906 *pr*97907 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)97908 **CHANGE HISTORY**

97909 First released in Issue 2.

97910 **Issue 5**97911 The option `[-f type]` is added to the SYNOPSIS. The option descriptions are presented in
97912 alphabetic order. The description of `-bt` is changed to “Number only non-empty lines”.97913 **Issue 6**97914 The obsolescent behavior allowing the options to be intermingled with the optional *file* operand
97915 is removed.97916 **Issue 7**

97917 Austin Group Interpretation 1003.1-2001 #092 is applied.

97918 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Utilities

nm

97919 NAME

97920 nm — write the name list of an object file (DEVELOPMENT)

97921 SYNOPSIS

97922 SD nm [-APv] [-g|-u] [-t *format*] *file*...

97923 XSI nm [-APv] [-efox] [-g|-u] [-t *format*] *file*...

97924 DESCRIPTION

97925 The *nm* utility shall display symbolic information appearing in the object file, executable file, or
 97926 object-file library named by *file*. If no symbolic information is available for a valid input file, the
 97927 *nm* utility shall report that fact, but not consider it an error condition.

97928 XSI The default base used when numeric values are written is unspecified. On XSI-conformant
 97929 systems, it shall be decimal.

97930 OPTIONS

97931 The *nm* utility shall conform to XBD Section 12.2 (on page 215).

97932 The following options shall be supported:

- 97933 **-A** Write the full pathname or library name of an object on each line.
- 97934 XSI **-e** Write only external (global) and static symbol information.
- 97935 XSI **-f** Produce full output. Write redundant symbols (*.text*, *.data*, and *.bss*), normally
 97936 suppressed.
- 97937 **-g** Write only external (global) symbol information.
- 97938 XSI **-o** Write numeric values in octal (equivalent to **-t o**).
- 97939 **-P** Write information in a portable output format, as specified in the STDOUT section.
- 97940 **-t *format*** Write each numeric value in the specified format. The format shall be dependent
 97941 on the single character used as the *format* option-argument:
 - 97942 XSI d The offset is written in decimal (default).
 - 97943 o The offset is written in octal.
 - 97944 x The offset is written in hexadecimal.
- 97945 **-u** Write only undefined symbols.
- 97946 **-v** Sort output by value instead of alphabetically.
- 97947 XSI **-x** Write numeric values in hexadecimal (equivalent to **-t x**).

97948 OPERANDS

97949 The following operand shall be supported:

97950 *file* A pathname of an object file, executable file, or object-file library.

97951 STDIN

97952 See the INPUT FILES section.

97953 INPUT FILES

97954 The input file shall be an object file, an object-file library whose format is the same as those
 97955 produced by the *ar* utility for link editing, or an executable file. The *nm* utility may accept
 97956 additional implementation-defined object library formats for the input file.

97957 **ENVIRONMENT VARIABLES**97958 The following environment variables shall affect the execution of *nm*:

- 97959 *LANG* Provide a default value for the internationalization variables that are unset or null.
 97960 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 97961 variables used to determine the values of locale categories.)
- 97962 *LC_ALL* If set to a non-empty string value, override the values of all the other
 97963 internationalization variables.
- 97964 *LC_COLLATE*
 97965 Determine the locale for character collation information for the symbol-name and
 97966 symbol-value collation sequences.
- 97967 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 97968 characters (for example, single-byte as opposed to multi-byte characters in
 97969 arguments).
- 97970 *LC_MESSAGES*
 97971 Determine the locale that should be used to affect the format and contents of
 97972 diagnostic messages written to standard error.
- 97973 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

97974 **ASYNCHRONOUS EVENTS**

97975 Default.

97976 **STDOUT**

97977 If symbolic information is present in the input files, then for each file or for each member of an
 97978 archive, the *nm* utility shall write the following information to standard output. By default, the
 97979 format is unspecified, but the output shall be sorted alphabetically by symbol name:

- 97980 • Library or object name, if *-A* is specified
- 97981 • Symbol name
- 97982 • Symbol type, which shall either be one of the following single characters or an
 97983 implementation-defined type represented by a single character:
 - 97984 A Global absolute symbol.
 - 97985 a Local absolute symbol.
 - 97986 B Global “bss” (that is, uninitialized data space) symbol.
 - 97987 b Local bss symbol.
 - 97988 D Global data symbol.
 - 97989 d Local data symbol.
 - 97990 T Global text symbol.
 - 97991 t Local text symbol.
 - 97992 U Undefined symbol.
- 97993 • Value of the symbol
- 97994 • The size associated with the symbol, if applicable

97995 This information may be supplemented by additional information specific to the
 97996 implementation.

Utilities

nm

If the **-P** option is specified, the previous information shall be displayed using the following portable format. The three versions differ depending on whether **-t d**, **-t o**, or **-t x** was specified, respectively:

```
"%s%s %s %d %d\n", <library/object name>, <name>, <type>,
    <value>, <size>
```

```
"%s%s %s %o %o\n", <library/object name>, <name>, <type>,
    <value>, <size>
```

```
"%s%s %s %x %x\n", <library/object name>, <name>, <type>,
    <value>, <size>
```

where *<library/object name>* shall be formatted as follows:

- If **-A** is not specified, *<library/object name>* shall be an empty string.
- If **-A** is specified and the corresponding *file* operand does not name a library:


```
"%s: ", <file>
```
- If **-A** is specified and the corresponding *file* operand names a library. In this case, *<object file>* shall name the object file in the library containing the symbol being described:


```
"%s[%s]: ", <file>, <object file>
```

If **-A** is not specified, then if more than one *file* operand is specified or if only one *file* operand is specified and it names a library, **nm** shall write a line identifying the object containing the following symbols before the lines containing those symbols, in the form:

- If the corresponding *file* operand does not name a library:


```
"%s:\n", <file>
```
- If the corresponding *file* operand names a library; in this case, *<object file>* shall be the name of the file in the library containing the following symbols:


```
"%s[%s]:\n", <file>, <object file>
```

If **-P** is specified, but **-t** is not, the format shall be as if **-t x** had been specified.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

- 0 Successful completion.
- >0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Mechanisms for dynamic linking make this utility less meaningful when applied to an executable file because a dynamically linked executable may omit numerous library routines that would be found in a statically linked executable.

EXAMPLES

None.

RATIONALE

Historical implementations of *nm* have used different bases for numeric output and supplied different default types of symbols that were reported. The *-t format* option, similar to that used in *od* and *strings*, can be used to specify the numeric base; *-g* and *-u* can be used to restrict the amount of output or the types of symbols included in the output.

The compromise of using *-t format* versus using *-d*, *-o*, and other similar options was necessary because of differences in the meaning of *-o* between implementations. The *-o* option from BSD has been provided here as *-A* to avoid confusion with the *-o* from System V (which has been provided here as *-t* and as *-o* on XSI-conformant systems).

The option list was significantly reduced from that provided by historical implementations.

The *nm* description is a subset of both the System V and BSD *nm* utilities with no specified default output.

It was recognized that mechanisms for dynamic linking make this utility less meaningful when applied to an executable file (because a dynamically linked executable file may omit numerous library routines that would be found in a statically linked executable file), but the value of *nm* during software development was judged to outweigh other limitations.

The default output format of *nm* is not specified because of differences in historical implementations. The *-P* option was added to allow some type of portable output format. After a comparison of the different formats used in SunOS, BSD, SVR3, and SVR4, it was decided to create one that did not match the current format of any of these four systems. The format devised is easy to parse by humans, easy to parse in shell scripts, and does not need to vary depending on locale (because no English descriptions are included). All of the systems currently have the information available to use this format.

The format given in *nm* STDOUT uses <space> characters between the fields, which may be any number of <blank> characters required to align the columns. The single-character types were selected to match historical practice, and the requirement that implementation additions also be single characters made parsing the information easier for shell scripts.

FUTURE DIRECTIONS

None.

SEE ALSO

ar, *c99*

XBD Chapter 8 (on page 173), Section 12.2 (on page 215)

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as supported when both the User Portability Utilities option and the Software Development Utilities option are supported.

98077 **Issue 7**

98078 The *nm* utility is removed from the User Portability Utilities option. User Portability Utilities is
98079 now an option for interactive utilities.

98080 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

nohup

Utilities

98081 **NAME**

98082 nohup — invoke a utility immune to hangups

98083 **SYNOPSIS**98084 nohup *utility* [*argument...*]98085 **DESCRIPTION**

98086 The *nohup* utility shall invoke the utility named by the *utility* operand with arguments supplied
 98087 as the *argument* operands. At the time the named *utility* is invoked, the SIGHUP signal shall be
 98088 set to be ignored.

98089 If standard input is associated with a terminal, the *nohup* utility may redirect standard input
 98090 from an unspecified file.

98091 If the standard output is a terminal, all output written by the named *utility* to its standard output
 98092 shall be appended to the end of the file **nohup.out** in the current directory. If **nohup.out** cannot
 98093 be created or opened for appending, the output shall be appended to the end of the file
 98094 **nohup.out** in the directory specified by the *HOME* environment variable. If neither file can be
 98095 created or opened for appending, *utility* shall not be invoked. If a file is created, the file's
 98096 permission bits shall be set to S_IRUSR | S_IWUSR.

98097 If standard error is a terminal and standard output is open but is not a terminal, all output
 98098 written by the named utility to its standard error shall be redirected to the same open file
 98099 description as the standard output. If standard error is a terminal and standard output either is a
 98100 terminal or is closed, the same output shall instead be appended to the end of the **nohup.out** file
 98101 as described above.

98102 **OPTIONS**

98103 None.

98104 **OPERANDS**

98105 The following operands shall be supported:

98106 *utility* The name of a utility that is to be invoked. If the *utility* operand names any of the
 98107 special built-in utilities in [Section 2.14](#) (on page 2334), the results are undefined.

98108 *argument* Any string to be supplied as an argument when invoking the utility named by the
 98109 *utility* operand.

98110 **STDIN**

98111 Not used.

98112 **INPUT FILES**

98113 None.

98114 **ENVIRONMENT VARIABLES**98115 The following environment variables shall affect the execution of *nohup*:

98116 *HOME* Determine the pathname of the user's home directory: if the output file **nohup.out**
 98117 cannot be created in the current directory, the *nohup* utility shall use the directory
 98118 named by *HOME* to create the file.

98119 *LANG* Provide a default value for the internationalization variables that are unset or null.
 98120 (See [XBD Section 8.2](#) (on page 174) for the precedence of internationalization
 98121 variables used to determine the values of locale categories.)

98122 *LC_ALL* If set to a non-empty string value, override the values of all the other
 98123 internationalization variables.

Utilities

nohup

98124	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
98125		
98126		
98127	<i>LC_MESSAGES</i>	
98128		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
98129		
98130	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
98131	<i>PATH</i>	Determine the search path that is used to locate the utility to be invoked. See XBD Chapter 8 (on page 173).
98132		
98133	ASYNCHRONOUS EVENTS	
98134		The <i>nohup</i> utility shall take the standard action for all signals except that SIGHUP shall be ignored.
98135		
98136	STDOUT	
98137		If the standard output is not a terminal, the standard output of <i>nohup</i> shall be the standard output generated by the execution of the <i>utility</i> specified by the operands. Otherwise, nothing shall be written to the standard output.
98138		
98139		
98140	STDERR	
98141		If the standard output is a terminal, a message shall be written to the standard error, indicating the name of the file to which the output is being appended. The name of the file shall be either nohup.out or \$HOME/nohup.out .
98142		
98143		
98144	OUTPUT FILES	
98145		Output written by the named utility is appended to the file nohup.out (or \$HOME/nohup.out), if the conditions hold as described in the DESCRIPTION .
98146		
98147	EXTENDED DESCRIPTION	
98148		None.
98149	EXIT STATUS	
98150		The following exit values shall be returned:
98151	126	The utility specified by <i>utility</i> was found but could not be invoked.
98152	127	An error occurred in the <i>nohup</i> utility or the utility specified by <i>utility</i> could not be found.
98153		
98154		Otherwise, the exit status of <i>nohup</i> shall be that of the utility specified by the <i>utility</i> operand.
98155	CONSEQUENCES OF ERRORS	
98156		Default.
98157	APPLICATION USAGE	
98158		The <i>command</i> , <i>env</i> , <i>nice</i> , <i>nohup</i> , <i>time</i> , and <i>xargs</i> utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish “failure to find a utility” from “invoked utility exited with an error indication”. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for “normal error conditions” and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to <i>exec</i> the utility fail with [ENOENT] , and uses 126 when any attempt to <i>exec</i> the utility fails for any other reason.
98159		
98160		
98161		
98162		
98163		
98164		
98165		
98166		
98167		

nohup*Utilities***EXAMPLES**

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done by placing pipelines and command lists in a single file; this file can then be invoked as a utility, and the *nohup* applies to everything in the file.

Alternatively, the following command can be used to apply *nohup* to a complex command:

```
nohup sh -c 'complex-command-line' </dev/null
```

RATIONALE

The 4.3 BSD version ignores SIGTERM and SIGHUP, and if *./nohup.out* cannot be used, it fails instead of trying to use *\$HOME/nohup.out*.

The *cs* utility has a built-in version of *nohup* that acts differently from the *nohup* defined in this volume of POSIX.1-2008.

The term *utility* is used, rather than *command*, to highlight the fact that shell compound commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility* includes user application programs and shell scripts, not just the standard utilities.

Historical versions of the *nohup* utility use default file creation semantics. Some more recent versions use the permissions specified here as an added security precaution.

Some historical implementations ignore SIGQUIT in addition to SIGHUP; others ignore SIGTERM. An early proposal allowed, but did not require, SIGQUIT to be ignored. Several reviewers objected that *nohup* should only modify the handling of SIGHUP as required by this volume of POSIX.1-2008.

Historical versions of *nohup* did not affect standard input, but that causes problems in the common scenario where the user logs into a system, types the command:

```
nohup make &
```

at the prompt, and then logs out. If standard input is not affected by *nohup*, the login session may not terminate for quite some time, since standard input remains open until *make* exits. To avoid this problem, POSIX.1-2008 allows implementations to redirect standard input if it is a terminal. Since the behavior is implementation-defined, portable applications that may run into the problem should redirect standard input themselves. For example, instead of:

```
nohup make &
```

an application can invoke:

```
nohup make </dev/null &
```

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2 (on page 2297), *sh*

XBD Chapter 8 (on page 173)

XSH *signal()*

CHANGE HISTORY

First released in Issue 2.

98207 **Issue 7**
98208

Austin Group Interpretations 1003.1-2001 #104, #105, and #106 are applied.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

od

Utilities

98209 **NAME**98210 **od** — dump files in various formats98211 **SYNOPSIS**

98212 **od** [**-v**] [**-A** *address_base*] [**-j** *skip*] [**-N** *count*] [**-t** *type_string*]...
 98213 [*file*...]

98214 XSI **od** [**-bcdosx**] [*file*] [[**+**] *offset* [**.**] [**b**]]

98215 **DESCRIPTION**

98216 The *od* utility shall write the contents of its input files to standard output in a user-specified
 98217 format.

98218 **OPTIONS**

98219 The *od* utility shall conform to XBD Section 12.2 (on page 215), except that the order of
 98220 XSI presentation of the **-t** options and the **-bcdosx** options is significant.

98221 The following options shall be supported:

98222 **-A** *address_base*

98223 Specify the input offset base. See the EXTENDED DESCRIPTION section. The
 98224 application shall ensure that the *address_base* option-argument is a character. The
 98225 characters 'd', 'o', and 'x' specify that the offset base shall be written in
 98226 decimal, octal, or hexadecimal, respectively. The character 'n' specifies that the
 98227 offset shall not be written.

98228 XSI **-b** Interpret bytes in octal. This shall be equivalent to **-t o1**.

98229 XSI **-c** Interpret bytes as characters specified by the current setting of the *LC_CTYPE*
 98230 category. Certain non-graphic characters appear as C escapes: "NUL=\0",
 98231 "BS=\b", "FF=\f", "NL=\n", "CR=\r", "HT=\t"; others appear as 3-digit octal
 98232 numbers.

98233 XSI **-d** Interpret words (two-byte units) in unsigned decimal. This shall be equivalent to
 98234 **-t u2**.

98235 **-j** *skip* Jump over *skip* bytes from the beginning of the input. The *od* utility shall read or
 98236 seek past the first *skip* bytes in the concatenated input files. If the combined input is
 98237 not at least *skip* bytes long, the *od* utility shall write a diagnostic message to
 98238 standard error and exit with a non-zero exit status.

98239 By default, the *skip* option-argument shall be interpreted as a decimal number.
 98240 With a leading 0x or 0X, the offset shall be interpreted as a hexadecimal number;
 98241 otherwise, with a leading '0', the offset shall be interpreted as an octal number.
 98242 Appending the character 'b', 'k', or 'm' to offset shall cause it to be interpreted
 98243 as a multiple of 512, 1024, or 1048576 bytes, respectively. If the *skip* number is
 98244 hexadecimal, any appended 'b' shall be considered to be the final hexadecimal
 98245 digit.

98246 **-N** *count* Format no more than *count* bytes of input. By default, *count* shall be interpreted as
 98247 a decimal number. With a leading 0x or 0X, *count* shall be interpreted as a
 98248 hexadecimal number; otherwise, with a leading '0', it shall be interpreted as an
 98249 octal number. If *count* bytes of input (after successfully skipping, if **-j** *skip* is
 98250 specified) are not available, it shall not be considered an error; the *od* utility shall
 98251 format the input that is available.

Utilities

od

98252	XSI	-o	Interpret <i>words</i> (two-byte units) in octal. This shall be equivalent to -t o2 .
98253	XSI	-s	Interpret <i>words</i> (two-byte units) in signed decimal. This shall be equivalent to
98254		-t d2 .	
98255		-t type_string	
98256			Specify one or more output types. See the EXTENDED DESCRIPTION section. The
98257			application shall ensure that the <i>type_string</i> option-argument is a string specifying
98258			the types to be used when writing the input data. The string shall consist of the
98259			type specification characters <i>a</i> , <i>c</i> , <i>d</i> , <i>f</i> , <i>o</i> , <i>u</i> , and <i>x</i> , specifying named character,
98260			character, signed decimal, floating point, octal, unsigned decimal, and
98261			hexadecimal, respectively. The type specification characters <i>d</i> , <i>f</i> , <i>o</i> , <i>u</i> , and <i>x</i> can be
98262			followed by an optional unsigned decimal integer that specifies the number of
98263			bytes to be transformed by each instance of the output type. The type specification
98264			character <i>f</i> can be followed by an optional <i>F</i> , <i>D</i> , or <i>L</i> indicating that the conversion
98265			should be applied to an item of type float , double , or long double , respectively.
98266			The type specification characters <i>d</i> , <i>o</i> , <i>u</i> , and <i>x</i> can be followed by an optional <i>C</i> , <i>S</i> ,
98267			<i>I</i> , or <i>L</i> indicating that the conversion should be applied to an item of type char ,
98268			short , int , or long , respectively. Multiple types can be concatenated within the
98269			same <i>type_string</i> and multiple -t options can be specified. Output lines shall be
98270			written for each type specified in the order in which the type specification
98271			characters are specified.
98272		-v	Write all input data. Without the -v option, any number of groups of output lines,
98273			which would be identical to the immediately preceding group of output lines
98274			(except for the byte offsets), shall be replaced with a line containing only an
98275			<asterisk> (' * ').
98276	XSI	-x	Interpret <i>words</i> (two-byte units) in hexadecimal. This shall be equivalent to -t x2 .
98277	XSI		Multiple types can be specified by using multiple -bcdostx options. Output lines are written for
98278			each type specified in the order in which the types are specified.
98279		OPERANDS	
98280			The following operands shall be supported:
98281		<i>file</i>	A pathname of a file to be read. If no <i>file</i> operands are specified, the standard input
98282			shall be used.
98283			If there are no more than two operands, none of the -A , -j , -N , -t , or -v options is
98284			specified, and either of the following is true: the first character of the last operand
98285			is a <plus-sign> (' + '), or there are two operands and the first character of the last
98286	XSI		operand is numeric; the last operand shall be interpreted as an offset operand on
98287			XSI-conformant systems. Under these conditions, the results are unspecified on
98288			systems that are not XSI-conformant systems.
98289	XSI	[+]<i>offset</i>[.][<i>b</i>]	The <i>offset</i> operand specifies the offset in the file where dumping is to commence.
98290			This operand is normally interpreted as octal bytes. If ' . ' is appended, the offset
98291			shall be interpreted in decimal. If ' b ' is appended, the offset shall be interpreted
98292			in units of 512 bytes.
98293		STDIN	
98294			The standard input shall be used if no <i>file</i> operands are specified, and shall be used if a <i>file</i>
98295			operand is ' - ' and the implementation treats the ' - ' as meaning standard input. Otherwise,
98296			the standard input shall not be used. See the INPUT FILES section.

98297 **INPUT FILES**

98298 The input files can be any file type.

98299 **ENVIRONMENT VARIABLES**98300 The following environment variables shall affect the execution of *od*:

98301 **LANG** Provide a default value for the internationalization variables that are unset or null.
 98302 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 98303 variables used to determine the values of locale categories.)

98304 **LC_ALL** If set to a non-empty string value, override the values of all the other
 98305 internationalization variables.

98306 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 98307 characters (for example, single-byte as opposed to multi-byte characters in
 98308 arguments and input files).

98309 **LC_MESSAGES**

98310 Determine the locale that should be used to affect the format and contents of
 98311 diagnostic messages written to standard error.

98312 **LC_NUMERIC**

98313 Determine the locale for selecting the radix character used when writing floating-
 98314 point formatted output.

98315 **XSI NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

98316 **ASYNCHRONOUS EVENTS**

98317 Default.

98318 **STDOUT**

98319 See the EXTENDED DESCRIPTION section.

98320 **STDERR**

98321 The standard error shall be used only for diagnostic messages.

98322 **OUTPUT FILES**

98323 None.

98324 **EXTENDED DESCRIPTION**

98325 The *od* utility shall copy sequentially each input file to standard output, transforming the input
 98326 **XSI** data according to the output types specified by the **-t** option or the **-bcdosx** options. If no
 98327 output type is specified, the default output shall be as if **-t oS** had been specified.

98328 The number of bytes transformed by the output type specifier *c* may be variable depending on
 98329 the **LC_CTYPE** category.

98330 The default number of bytes transformed by output type specifiers *d*, *f*, *o*, *u*, and *x* corresponds
 98331 to the various C-language types as follows. If the *c99* compiler is present on the system, these
 98332 specifiers shall correspond to the sizes used by default in that compiler. Otherwise, these sizes
 98333 may vary among systems that conform to POSIX.1-2008.

- 98334 • For the type specifier characters *d*, *o*, *u*, and *x*, the default number of bytes shall
- 98335 correspond to the size of the underlying implementation's basic integer type. For these
- 98336 specifier characters, the implementation shall support values of the optional number of
- 98337 bytes to be converted corresponding to the number of bytes in the C-language types **char**,
- 98338 **short**, **int**, and **long**. These numbers can also be specified by an application as the
- 98339 characters '*C*', '*S*', '*I*', and '*L*', respectively. The implementation shall also support
- 98340 the values 1, 2, 4, and 8, even if it provides no C-Language types of those sizes. The

implementation shall support the decimal value corresponding to the C-language type **long long**. The byte order used when interpreting numeric values is implementation-defined, but shall correspond to the order in which a constant of the corresponding type is stored in memory on the system.

- For the type specifier character $\mathfrak{f}$, the default number of bytes shall correspond to the number of bytes in the underlying implementation's basic double precision floating-point data type. The implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types **float**, **double**, and **long double**. These numbers can also be specified by an application as the characters 'F', 'D', and 'L', respectively.

The type specifier character **a** specifies that bytes shall be interpreted as named characters from the International Reference Version (IRV) of the ISO/IEC 646:1991 standard. Only the least significant seven bits of each byte shall be used for this type specification. Bytes with the values listed in the following table shall be written using the corresponding names for those characters.

Table 4-13 Named Characters in *od*

Value	Name	Value	Name	Value	Name	Value	Name
\000	nul	\001	soh	\002	stx	\003	etx
\004	eot	\005	enq	\006	ack	\007	bel
\010	bs	\011	ht	\012	If or nl*	\013	vt
\014	ff	\015	cr	\016	so	\017	si
\020	dle	\021	dc1	\022	dc2	\023	dc3
\024	dc4	\025	nak	\026	syn	\027	etb
\030	can	\031	em	\032	sub	\033	esc
\034	fs	\035	gs	\036	rs	\037	us
\040	sp	\177	del				

Note: The "\012" value may be written either as **If** or **nl**.

The type specifier character **c** specifies that bytes shall be interpreted as characters specified by the current setting of the `LC_CTYPE` locale category. Characters listed in the table in XBD Chapter 5 (on page 121) ('', '\a', '\b', '\f', '\n', '\r', '\t', '\v') shall be written as the corresponding escape sequences, except that <backslash> shall be written as a single <backslash> and a NUL shall be written as '\0'. Other non-printable characters shall be written as one three-digit octal number for each byte in the character. Printable multi-byte characters shall be written in the area corresponding to the first byte of the character; the two-character sequence "***" shall be written in the area corresponding to each remaining byte in the character, as an indication that the character is continued. When either the **-j skip** or **-N count** option is specified along with the **c** type specifier, and this results in an attempt to start or finish in the middle of a multi-byte character, the result is implementation-defined.

The input data shall be manipulated in blocks, where a block is defined as a multiple of the least common multiple of the number of bytes transformed by the specified output types. If the least common multiple is greater than 16, the results are unspecified. Each input block shall be written as transformed by each output type, one per written line, in the order that the output types were specified. If the input block size is larger than the number of bytes transformed by the output type, the output type shall sequentially transform the parts of the input block, and the output from each of the transformations shall be separated by one or more <blank> characters.

If, as a result of the specification of the **-N** option or end-of-file being reached on the last input file, input data only partially satisfies an output type, the input shall be extended sufficiently

98388 with null bytes to write the last byte of the input.

98389 Unless **-A n** is specified, the first output line produced for each input block shall be preceded by
 98390 the input offset, cumulative across input files, of the next byte to be written. The format of the
 98391 input offset is unspecified; however, it shall not contain any <blank> characters, shall start at the
 98392 first character of the output line, and shall be followed by one or more <blank> characters. In
 98393 addition, the offset of the byte following the last byte written shall be written after all the input
 98394 data has been processed, but shall not be followed by any <blank> characters.

98395 If no **-A** option is specified, the input offset base is unspecified.

98396 EXIT STATUS

98397 The following exit values shall be returned:

98398 0 All input files were processed successfully.

98399 >0 An error occurred.

98400 CONSEQUENCES OF ERRORS

98401 Default.

98402 APPLICATION USAGE

98403 XSI-conformant applications are warned not to use filenames starting with '+' or a first
 98404 operand starting with a numeric character so that the old functionality can be maintained by
 98405 implementations, unless they specify one of the **-A**, **-j**, or **-N** options. To guarantee that one of
 98406 these filenames is always interpreted as a filename, an application could always specify the
 98407 address base format with the **-A** option.

98408 EXAMPLES

98409 If a file containing 128 bytes with decimal values zero to 127, in increasing order, is supplied as
 98410 standard input to the command:

98411 `od -A d -t a`

98412 on an implementation using an input block size of 16 bytes, the standard output, independent of
 98413 the current locale setting, would be similar to:

```

98414 00000000 nul soh stx etx eot enq ack bel bs ht nl vt ff cr so si
98415 00000016 dle dc1 dc2 dc3 dc4 nak syn etb can em sub esc fs gs rs us
98416 00000032 sp " # $ % & ' ( ) * + , - . /
98417 00000048 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
98418 00000064 @ A B C D E F G H I J K L M N O
98419 00000080 P Q R S T U V W X Y Z [ \ ] ^ _
98420 00000096 ` a b c d e f g h i j k l m n o
98421 00000112 p q r s t u v w x y z { | } ~ del
98422 00000128
  
```

98423 Note that this volume of POSIX.1-2008 allows **nl** or **lf** to be used as the name for the
 98424 ISO/IEC 646:1991 standard IRV character with decimal value 10. The IRV names this character
 98425 **lf** (line feed), but traditional implementations have referred to this character as newline (**nl**) and
 98426 the POSIX locale character set symbolic name for the corresponding character is a <newline>.

98427 The command:

98428 `od -A o -t o2x2x -N 18`

98429 on a system with 32-bit words and an implementation using an input block size of 16 bytes
 98430 could write 18 bytes in approximately the following format:

98431 00000000 032056 031440 041123 042040 052516 044530 020043 031464

```

98432          342e  3320  4253  4420  554e  4958  2023  3334
98433          342e3320      42534420      554e4958      20233334
98434 0000020 032472
98435          353a
98436          353a0000
98437 0000022

```

The command:

```

98439 od -A d -t f -t o4 -t x4 -N 24 -j 0x15

```

on a system with 64-bit doubles (for example, IEEE Std 754-1985 double precision floating-point format) would skip 21 bytes of input data and then write 24 bytes in approximately the following format:

```

98443 0000000 1.000000000000000e+00 1.573500000000000e+01
98444 0777400000 0000000000 10013674121 35341217270
98445 3ff00000 00000000 402f3851 eb851eb8
98446 0000016 1.406682300000000e+02
98447 10030312542 04370303230
98448 40619562 23e18698
98449 0000024

```

RATIONALE

The *od* utility went through several names in early proposals, including *hd*, *xd*, and most recently *hexdump*. There were several objections to all of these based on the following reasons:

- The *hd* and *xd* names conflicted with historical utilities that behaved differently.
- The *hexdump* description was much more complex than needed for a simple dump utility.
- The *od* utility has been available on all historical implementations and there was no need to create a new name for a utility so similar to the historical *od* utility.

The original reasons for not standardizing historical *od* were also fairly widespread. Those reasons are given below along with rationale explaining why the standard developers believe that this version does not suffer from the indicated problem:

- The BSD and System V versions of *od* have diverged, and the intersection of features provided by both does not meet the needs of the user community. In fact, the System V version only provides a mechanism for dumping octal bytes and **shorts**, signed and unsigned decimal **shorts**, hexadecimal **shorts**, and ASCII characters. BSD added the ability to dump **floats**, **doubles**, named ASCII characters, and octal, signed decimal, unsigned decimal, and hexadecimal **longs**. The version presented here provides more normalized forms for dumping bytes, **shorts**, **ints**, and **longs** in octal, signed decimal, unsigned decimal, and hexadecimal; **float**, **double**, and **long double**; and named ASCII as well as current locale characters.
- It would not be possible to come up with a compatible superset of the BSD and System V flags that met the requirements of the standard developers. The historical default *od* output is the specified default output of this utility. None of the option letters chosen for this version of *od* conflict with any of the options to historical versions of *od*.
- On systems with different sizes for **short**, **int**, and **long**, there was no way to ask for dumps of **ints**, even in the BSD version. Because of the way options are named, the name space could not be extended to solve these problems. This is why the **-t** option was added (with type specifiers more closely matched to the *printf()* formats used in the rest of this volume of POSIX.1-2008) and the optional field sizes were added to the **d**, **f**, **o**, **u**, and **x** type

specifiers. It is also one of the reasons why the historical practice was not mandated as a required obsolescent form of *od*. (Although the old versions of *od* are not listed as an obsolescent form, implementations are urged to continue to recognize the older forms for several more years.) The *a*, *c*, *f*, *o*, and *x* types match the meaning of the corresponding format characters in the historical implementations of *od* except for the default sizes of the fields converted. The *d* format is signed in this volume of POSIX.1-2008 to match the *printf()* notation. (Historical versions of *od* used *d* as a synonym for *u* in this version. The System V implementation uses *s* for signed decimal; BSD uses *i* for signed decimal and *s* for null-terminated strings.) Other than *d* and *u*, all of the type specifiers match format characters in the historical BSD version of *od*.

The sizes of the C-language types **char**, **short**, **int**, **long**, **float**, **double**, and **long double** are used even though it is recognized that there may be zero or more than one compiler for the C language on an implementation and that they may use different sizes for some of these types. (For example, one compiler might use 2 bytes **shorts**, 2 bytes **ints**, and 4 bytes **longs**, while another compiler (or an option to the same compiler) uses 2 bytes **shorts**, 4 bytes **ints**, and 4 bytes **longs**.) Nonetheless, there has to be a basic size known by the implementation for these types, corresponding to the values reported by invocations of the *getconf* utility when called with *system_var* operands {UCHAR_MAX}, {USHORT_MAX}, {UINT_MAX}, and {ULONG_MAX} for the types **char**, **short**, **int**, and **long**, respectively. There are similar constants required by the ISO C standard, but not required by the System Interfaces volume of POSIX.1-2008 or this volume of POSIX.1-2008. They are {FLT_MANT_DIG}, {DBL_MANT_DIG}, and {LDBL_MANT_DIG} for the types **float**, **double**, and **long double**, respectively. If the optional *c99* utility is provided by the implementation and used as specified by this volume of POSIX.1-2008, these are the sizes that would be provided. If an option is used that specifies different sizes for these types, there is no guarantee that the *od* utility is able to interpret binary data output by such a program correctly.

This volume of POSIX.1-2008 requires that the numeric values of these lengths be recognized by the *od* utility and that symbolic forms also be recognized. Thus, a conforming application can always look at an array of **unsigned long** data elements using *od -t uL*.

- The method of specifying the format for the address field based on specifying a starting offset in a file unnecessarily tied the two together. The *-A* option now specifies the address base and the *-S* option specifies a starting offset.
- It would be difficult to break the dependence on US ASCII to achieve an internationalized utility. It does not seem to be any harder for *od* to dump characters in the current locale than it is for the *ed* or *sed* *l* commands. The *c* type specifier does this without difficulty and is completely compatible with the historical implementations of the *c* format character when the current locale uses a superset of the ISO/IEC 646: 1991 standard as a codeset. The *a* type specifier (from the BSD *a* format character) was left as a portable means to dump ASCII (or more correctly ISO/IEC 646: 1991 standard (IRV)) so that headers produced by *pax* could be deciphered even on systems that do not use the ISO/IEC 646: 1991 standard as a subset of their base codeset.

The use of "***" as an indication of continuation of a multi-byte character in *c* specifier output was chosen based on seeing an implementation that uses this method. The continuation bytes have to be marked in a way that is not ambiguous with another single-byte or multi-byte character.

An early proposal used *-S* and *-n*, respectively, for the *-j* and *-N* options eventually selected. These were changed to avoid conflicts with historical implementations.

98527 The original standard specified **-t o2** as the default when no output type was given. This was
 98528 changed to **-t oS** (the length of a **short**) to accommodate a supercomputer implementation that
 98529 historically used 64 bits as its default (and that defined shorts as 64 bits). This change should not
 98530 affect conforming applications. The requirement to support lengths of 1, 2, and 4 was added at
 98531 the same time to address an historical implementation that had no two-byte data types in its C
 98532 compiler.

98533 The use of a basic integer data type is intended to allow the implementation to choose a word
 98534 size commonly used by applications on that architecture.

98535 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
 98536 but this has been modified in this version.

98537 FUTURE DIRECTIONS

98538 All option and operand interfaces marked XSI may be removed in a future version.

98539 SEE ALSO

98540 *c99*, *sed*

98541 XBD [Chapter 5](#) (on page 121), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)

98542 CHANGE HISTORY

98543 First released in Issue 2.

98544 Issue 5

98545 In the description of the **-c** option, the phrase “This is equivalent to **-t c.**” is deleted.

98546 The FUTURE DIRECTIONS section is modified.

98547 Issue 6

98548 The *od* utility is changed to remove the assumption that **short** was a two-byte entity, as per the
 98549 revisions in the IEEE P1003.2b draft standard.

98550 The normative text is reworded to avoid use of the term “must” for application requirements.

98551 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/33 is applied, correcting the examples
 98552 which used an undefined **-n** option, which should have been **-N**.

98553 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/19 is applied, removing text describing
 98554 behavior on systems with bytes consisting of more than eight bits.

98555 Issue 7

98556 Austin Group Interpretation 1003.1-2001 #092 is applied.

98557 SD5-XCU-ERN-37 is applied, updating the OPERANDS section.

98558 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

paste*Utilities*98559 **NAME**98560 **paste** — merge corresponding or subsequent lines of files98561 **SYNOPSIS**98562 **paste** [-s] [-d *list*] *file*...98563 **DESCRIPTION**98564 The *paste* utility shall concatenate the corresponding lines of the given input files, and write the
98565 resulting lines to standard output.98566 The default operation of *paste* shall concatenate the corresponding lines of the input files. The
98567 <newline> of every line except the line from the last input file shall be replaced with a <tab>.98568 If an end-of-file condition is detected on one or more input files, but not all input files, *paste* shall
98569 behave as though empty lines were read from the files on which end-of-file was detected, unless
98570 the -s option is specified.98571 **OPTIONS**98572 The *paste* utility shall conform to XBD [Section 12.2](#) (on page 215).

98573 The following options shall be supported:

98574 **-d list** Unless a <backslash> character appears in *list*, each character in *list* is an element
98575 specifying a delimiter character. If a <backslash> character appears in *list*, the
98576 <backslash> character and one or more characters following it are an element
98577 specifying a delimiter character as described below. These elements specify one or
98578 more delimiters to use, instead of the default <tab>, to replace the <newline> of
98579 the input lines. The elements in *list* shall be used circularly; that is, when the list is
98580 exhausted the first element from the list is reused. When the -s option is specified:

- 98581
- The last <newline> in a file shall not be modified.
 - The delimiter shall be reset to the first element of *list* after each *file* operand is processed.

98584 When the -s option is not specified:

- 98585
- The <newline> characters in the file specified by the last *file* operand shall not be modified.
 - The delimiter shall be reset to the first element of *list* each time a line is processed from each file.

98589 If a <backslash> character appears in *list*, it and the character following it shall be
98590 used to represent the following delimiter characters:

98591 \n <newline>.

98592 \t <tab>.

98593 \\ <backslash> character.

98594 \0 Empty string (not a null character). If '\0' is immediately followed by the
98595 character 'x', the character 'X', or any character defined by the LC_CTYPE
98596 **digit** keyword (see XBD [Chapter 7](#), on page 135), the results are unspecified.

98597 If any other characters follow the <backslash>, the results are unspecified.

98598 **-s** Concatenate all of the lines of each separate input file in command line order. The
98599 <newline> of every line except the last line in each input file shall be replaced with
98600 the <tab>, unless otherwise specified by the -d option.

Utilities

paste

98601 **OPERANDS**

98602 The following operand shall be supported:

98603 *file* A pathname of an input file. If '-' is specified for one or more of the *files*, the
 98604 standard input shall be used; the standard input shall be read one line at a time,
 98605 circularly, for each instance of '-'. Implementations shall support pasting of at
 98606 least 12 *file* operands.

98607 **STDIN**

98608 The standard input shall be used only if one or more *file* operands is '-'. See the INPUT FILES
 98609 section.

98610 **INPUT FILES**

98611 The input files shall be text files, except that line lengths shall be unlimited.

98612 **ENVIRONMENT VARIABLES**

98613 The following environment variables shall affect the execution of *paste*:

98614 *LANG* Provide a default value for the internationalization variables that are unset or null.
 98615 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables
 98616 used to determine the values of locale categories.)

98617 *LC_ALL* If set to a non-empty string value, override the values of all the other
 98618 internationalization variables.

98619 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 98620 characters (for example, single-byte as opposed to multi-byte characters in
 98621 arguments and input files).

98622 *LC_MESSAGES*

98623 Determine the locale that should be used to affect the format and contents of
 98624 diagnostic messages written to standard error.

98625 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

98626 **ASYNCHRONOUS EVENTS**

98627 Default.

98628 **STDOUT**

98629 Concatenated lines of input files shall be separated by the <tab> (or other characters under the
 98630 control of the -d option) and terminated by a <newline>.

98631 **STDERR**

98632 The standard error shall be used only for diagnostic messages.

98633 **OUTPUT FILES**

98634 None.

98635 **EXTENDED DESCRIPTION**

98636 None.

98637 **EXIT STATUS**

98638 The following exit values shall be returned:

98639 0 Successful completion.

98640 >0 An error occurred.

CONSEQUENCES OF ERRORS

If one or more input files cannot be opened when the `-s` option is not specified, a diagnostic message shall be written to standard error, but no output is written to standard output. If the `-s` option is specified, the *paste* utility shall provide the default behavior described in [Section 1.4](#) (on page 2288).

APPLICATION USAGE

When the escape sequences of the *list* option-argument are used in a shell script, they must be quoted; otherwise, the shell treats the `<backslash>` as a special character.

Conforming applications should only use the specific `<backslash>`-escaped delimiters presented in this volume of POSIX.1-2008. Historical implementations treat `'\x'`, where `'x'` is not in this list, as `'x'`, but future implementations are free to expand this list to recognize other common escapes similar to those accepted by *printf* and other standard utilities.

Most of the standard utilities work on text files. The *cut* utility can be used to turn files with arbitrary line lengths into a set of text files containing the same data. The *paste* utility can be used to create (or recreate) files with arbitrary line lengths. For example, if *file* contains long lines:

```
cut -b 1-500 -n file > file1
cut -b 501- -n file > file2
```

creates **file1** (a text file) with lines no longer than 500 bytes (plus the `<newline>`) and **file2** that contains the remainder of the data from *file*. Note that **file2** is not a text file if there are lines in *file* that are longer than `500 + {LINE_MAX}` bytes. The original file can be recreated from **file1** and **file2** using the command:

```
paste -d "\0" file1 file2 > file
```

The commands:

```
paste -d "\0" ...
paste -d "" ...
```

are not necessarily equivalent; the latter is not specified by this volume of POSIX.1-2008 and may result in an error. The construct `'\0'` is used to mean “no separator” because historical versions of *paste* did not follow the syntax guidelines, and the command:

```
paste -d "" ...
```

could not be handled properly by *getopt()*.

EXAMPLES

1. Write out a directory in four columns:

```
ls | paste - - - -
```

2. Combine pairs of lines from a file into single lines:

```
paste -s -d "\t\n" file
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

98680 **SEE ALSO**98681 [Section 1.4](#) (on page 2288), *cut*, *grep*, *pr*98682 XBD [Chapter 7](#) (on page 135), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)98683 **CHANGE HISTORY**

98684 First released in Issue 2.

98685 **Issue 6**

98686 The normative text is reworded to avoid use of the term “must” for application requirements.

98687 **Issue 7**

98688 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

patch

Utilities

98689 **NAME**98690 **patch** — apply changes to files98691 **SYNOPSIS**

98692 **patch** **[-blNR]** **[-c|-e|-n|-u]** **[-d dir]** **[-D define]** **[-i patchfile]**
 98693 **[-o outfile]** **[-p num]** **[-r rejectfile]** **[file]**

98694 **DESCRIPTION**

98695 The *patch* utility shall read a source (patch) file containing any of four forms of difference (diff)
 98696 listings produced by the *diff* utility (normal, copied context, unified context, or in the style of *ed*)
 98697 and apply those differences to a file. By default, *patch* shall read from the standard input.

98698 The *patch* utility shall attempt to determine the type of the *diff* listing, unless overruled by a **-c**,
 98699 **-e**, **-n**, or **-u** option.

98700 If the patch file contains more than one patch, *patch* shall attempt to apply each of them as if they
 98701 came from separate patch files. (In this case, the application shall ensure that the name of the
 98702 patch file is determinable for each *diff* listing.)

98703 **OPTIONS**

98704 The *patch* utility shall conform to XBD [Section 12.2](#) (on page 215).

98705 The following options shall be supported:

- 98706 **-b** Save a copy of the original contents of each modified file, before the differences are
 98707 applied, in a file of the same name with the suffix **.orig** appended to it. If the file
 98708 already exists, it shall be overwritten, if multiple patches are applied to the same
 98709 file, the **.orig** file shall be written only for the first patch. When the **-o outfile** option
 98710 is also specified, *file.orig* shall not be created but, if *outfile* already exists, *outfile.orig*
 98711 shall be created.
- 98712 **-c** Interpret the patch file as a copied context difference (the output of the utility *diff*
 98713 when the **-c** or **-C** options are specified).
- 98714 **-d dir** Change the current directory to *dir* before processing as described in the
 98715 EXTENDED DESCRIPTION section.
- 98716 **-D define** Mark changes with one of the following C preprocessor constructs:
- 98717 `#ifdef define`
 98718 `...`
 98719 `#endif`
- 98720 `#ifndef define`
 98721 `...`
 98722 `#endif`
- 98723 optionally combined with the C preprocessor construct **#else**. If the patched file is
 98724 processed with the C preprocessor, where the macro *define* is defined, the output
 98725 shall contain the changes from the patch file; otherwise, the output shall not
 98726 contain the patches specified in the patch file.
- 98727 **-e** Interpret the patch file as an *ed* script, rather than a *diff* script.
- 98728 **-i patchfile** Read the patch information from the file named by the pathname *patchfile*, rather
 98729 than the standard input.
- 98730 **-l** (The letter ell.) Cause any sequence of <blank> characters in the difference script to
 98731 match any sequence of <blank> characters in the input file. Other characters shall
 98732 be matched exactly.

Utilities

patch

98733	-n	Interpret the script as a normal difference.
98734	-N	Ignore patches where the differences have already been applied to the file; by default, already-applied patches shall be rejected.
98735		
98736	-o outfile	Instead of modifying the files (specified by the <i>file</i> operand or the difference listings) directly, write a copy of the file referenced by each patch, with the appropriate differences applied, to <i>outfile</i> . Multiple patches for a single file shall be applied to the intermediate versions of the file created by any previous patches, and shall result in multiple, concatenated versions of the file being written to <i>outfile</i> .
98737		
98738		
98739		
98740		
98741		
98742	-p num	For all pathnames in the patch file that indicate the names of files to be patched, delete <i>num</i> pathname components from the beginning of each pathname. If the pathname in the patch file is absolute, any leading <slash> characters shall be considered the first component (that is, -p 1 shall remove the leading <slash> characters). Specifying -p 0 shall cause the full pathname to be used. If -p is not specified, only the basename (the final pathname component) shall be used.
98743		
98744		
98745		
98746		
98747		
98748	-R	Reverse the sense of the patch script; that is, assume that the difference script was created from the new version to the old version. The -R option cannot be used with <i>ed</i> scripts. The <i>patch</i> utility shall attempt to reverse each portion of the script before applying it. Rejected differences shall be saved in swapped format. If this option is not specified, and until a portion of the patch file is successfully applied, <i>patch</i> attempts to apply each portion in its reversed sense as well as in its normal sense. If the attempt is successful, the user shall be prompted to determine whether the -R option should be set.
98749		
98750		
98751		
98752		
98753		
98754		
98755		
98756	-r rejectfile	Override the default reject filename. In the default case, the reject file shall have the same name as the output file, with the suffix .rej appended to it; see Patch Application (on page 2997).
98757		
98758		
98759	-u	Interpret the patch file as a unified context difference (the output of the <i>diff</i> utility when the -u or -U options are specified).
98760		

OPERANDS

The following operand shall be supported:

98763	<i>file</i>	A pathname of a file to patch.

STDIN

See the INPUT FILES section.

INPUT FILES

Input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *patch*:

98770	LANG	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) the precedence of internationalization variables used to determine the values of locale categories.)
98771		
98772		
98773	LC_ALL	If set to a non-empty string value, override the values of all the other internationalization variables.
98774		
98775	LC_COLLATE	Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the extended regular expression defined for
98776		
98777		

patch*Utilities*

98778		the yesexpr locale keyword in the <i>LC_MESSAGES</i> category.
98779	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), and the behavior of character classes used in the extended regular expression defined for the yesexpr locale keyword in the <i>LC_MESSAGES</i> category.
98780		
98781		
98782		
98783		
98784	<i>LC_MESSAGES</i>	
98785		Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of diagnostic messages and prompts written to standard error.
98786		
98787		
98788	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
98789	<i>LC_TIME</i>	Determine the locale for recognizing the format of file timestamps written by the <i>diff</i> utility in a context-difference input file.
98790		
98791	ASYNCHRONOUS EVENTS	
98792		Default.
98793	STDOUT	
98794		Not used.
98795	STDERR	
98796		The standard error shall be used for diagnostic and informational messages.
98797	OUTPUT FILES	
98798		The output of the <i>patch</i> utility, the save files (.orig suffixes), and the reject files (.rej suffixes) shall be text files.
98799		
98800	EXTENDED DESCRIPTION	
98801		A patch file may contain patching instructions for more than one file; filenames shall be determined as specified in Filename Determination (on page 2997). When the -b option is specified, for each patched file, the original shall be saved in a file of the same name with the suffix .orig appended to it.
98802		
98803		
98804		
98805		For each patched file, a reject file may also be created as noted in Patch Application (on page 2997). In the absence of a -r option, the name of this file shall be formed by appending the suffix .rej to the original filename.
98806		
98807		
98808	Patch File Format	
98809		The patch file shall contain zero or more lines of header information followed by one or more patches. Each patch shall contain zero or more lines of filename identification in the format produced by the -c , -C , -u , or -U options of the <i>diff</i> utility, and one or more sets of <i>diff</i> output, which are customarily called <i>hunks</i> .
98810		
98811		
98812		
98813		The <i>patch</i> utility shall recognize the following expression in the header information:
98814	Index: <i>pathname</i>	
98815		The file to be patched is named <i>pathname</i> .
98816		If all lines (including headers) within a patch begin with the same leading sequence of <blank> characters, the <i>patch</i> utility shall remove this sequence before proceeding. Within each patch, if the type of difference is common context, the <i>patch</i> utility shall recognize the following expressions:
98817		
98818		
98819		

98820 *** *filename timestamp*
 98821 The patches arose from *filename*.

98822 --- *filename timestamp*
 98823 The patches should be applied to *filename*.

98824 If the type of difference is unified context, the *patch* utility shall recognize the following
 98825 expressions:

98826 --- *filename timestamp*
 98827 The patches arose from *filename*.

98828 +++ *filename timestamp*
 98829 The patches should be applied to *filename*.

98830 Each hunk within a patch shall be the *diff* output to change a line range within the original file.
 98831 The line numbers for successive hunks within a patch shall occur in ascending order.

98832 Filename Determination

98833 If no *file* operand is specified, *patch* shall perform the following steps to determine the filename
 98834 to use:

- 98835 1. If the type of *diff* is context, the *patch* utility shall delete pathname components (as
 98836 specified by the **-p** option) from the filename on the line beginning with "***" (if copied
 98837 context) or "---" (if unified context), then test for the existence of this file relative to the
 98838 current directory (or the directory specified with the **-d** option). If the file exists, the *patch*
 98839 utility shall use this filename.
- 98840 2. If the type of *diff* is context, the *patch* utility shall delete the pathname components (as
 98841 specified by the **-p** option) from the filename on the line beginning with "---" (if copied
 98842 context) or "+++" (if unified context), then test for the existence of this file relative to the
 98843 current directory (or the directory specified with the **-d** option). If the file exists, the *patch*
 98844 utility shall use this filename.
- 98845 3. If the header information contains a line beginning with the string **Index:**, the *patch* utility
 98846 shall delete pathname components (as specified by the **-p** option) from this line, then test
 98847 for the existence of this file relative to the current directory (or the directory specified
 98848 with the **-d** option). If the file exists, the *patch* utility shall use this filename.
- 98849 XSI 4. If an **SCCS** directory exists in the current directory, *patch* shall attempt to perform a **get -e**
 98850 **SCCS/s.filename** command to retrieve an editable version of the file. If the file exists, the
 98851 *patch* utility shall use this filename.
- 98852 5. The *patch* utility shall write a prompt to standard output and request a filename
 98853 interactively from the controlling terminal (for example, **/dev/tty**).

98854 Patch Application

98855 If the **-c**, **-e**, **-n**, or **-u** option is present, the *patch* utility shall interpret information within each
 98856 hunk as a copied context difference, an *ed* difference, a normal difference, or a unified context
 98857 difference, respectively. In the absence of any of these options, the *patch* utility shall determine
 98858 the type of difference based on the format of information within the hunk.

98859 For each hunk, the *patch* utility shall begin to search for the place to apply the patch at the line
 98860 number at the beginning of the hunk, plus or minus any offset used in applying the previous
 98861 hunk. If lines matching the hunk context are not found, *patch* shall scan both forwards and
 98862 backwards at least 1 000 bytes for a set of lines that match the hunk context.

patch*Utilities*

98863 If no such place is found and it is a context difference, then another scan shall take place,
 98864 ignoring the first and last line of context. If that fails, the first two and last two lines of context
 98865 shall be ignored and another scan shall be made. Implementations may search more extensively
 98866 for installation locations.

98867 If no location can be found, the *patch* utility shall append the hunk to the reject file. A rejected
 98868 hunk that is a copied context difference, an *ed* difference, or a normal difference shall be written
 98869 in copied-context-difference format regardless of the format of the patch file. It is
 98870 implementation-defined whether a rejected hunk that is a unified context difference is written in
 98871 copied-context-difference format or in unified-context-difference format. If the input was a
 98872 normal or *ed*-style difference, the reject file may contain differences with zero lines of context.
 98873 The line numbers on the hunks in the reject file may be different from the line numbers in the
 98874 patch file since they shall reflect the approximate locations for the failed hunks in the new file
 98875 rather than the old one.

98876 If the type of patch is an *ed* diff, the implementation may accomplish the patching by invoking
 98877 the *ed* utility.

EXIT STATUS

98878 The following exit values shall be returned:

- 98880 0 Successful completion.
- 98881 1 One or more lines were written to a reject file.
- 98882 >1 An error occurred.

CONSEQUENCES OF ERRORS

98883 Patches that cannot be correctly placed in the file shall be written to a reject file.

APPLICATION USAGE

98886 The **-R** option does not work with *ed* scripts because there is too little information to reconstruct
 98887 the reverse operation.

98888 The **-p** option makes it possible to customize a patch file to local user directory structures
 98889 without manually editing the patch file. For example, if the filename in the patch file was:

98890 `/curds/whey/src/blurfl/blurfl.c`

98891 Setting **-p 0** gives the entire pathname unmodified; **-p 1** gives:

98892 `curds/whey/src/blurfl/blurfl.c`

98893 without the leading <slash>, **-p 4** gives:

98894 `blurfl/blurfl.c`

98895 and not specifying **-p** at all gives:

98896 `blurfl.c` .

EXAMPLES

98898 None.

RATIONALE

98900 Some of the functionality in historical *patch* implementations was not specified. The following
 98901 documents those features present in historical implementations that have not been specified.

98902 A deleted piece of functionality was the '+' pseudo-option allowing an additional set of options
 98903 and a patch file operand to be given. This was seen as being insufficiently useful to standardize.

98904 In historical implementations, if the string "Prereq:" appeared in the header, the *patch* utility

would search for the corresponding version information (the string specified in the header, delimited by <blank> characters or the beginning or end of a line or the file) anywhere in the original file. This was deleted as too simplistic and insufficiently trustworthy a mechanism to standardize. For example, if:

Prereq: 1.2

were in the header, the presence of a delimited 1.2 anywhere in the file would satisfy the prerequisite.

The following options were dropped from historical implementations of *patch* as insufficiently useful to standardize:

-b The **-b** option historically provided a method for changing the name extension of the backup file from the default **.orig**. This option has been modified and retained in this volume of POSIX.1-2008.

-F The **-F** option specified the number of lines of a context diff to ignore when searching for a place to install a patch.

-f The **-f** option historically caused *patch* not to request additional information from the user.

-r The **-r** option historically provided a method of overriding the extension of the reject file from the default **.rej**.

-s The **-s** option historically caused *patch* to work silently unless an error occurred.

-x The **-x** option historically set internal debugging flags.

In some file system implementations, the saving of a **.orig** file may produce unwanted results. In the case of 12, 13, or 14-character filenames (on file systems supporting 14-character maximum filenames), the **.orig** file overwrites the new file. The reject file may also exceed this filename limit. It was suggested, due to some historical practice, that a <tilde> ('~') suffix be used instead of **.orig** and some other character instead of the **.rej** suffix. This was rejected because it is not obvious to the user which file is which. The suffixes **.orig** and **.rej** are clearer and more understandable.

The **-b** option has the opposite sense in some historical implementations—do not save the **.orig** file. The default case here is not to save the files, making *patch* behave more consistently with the other standard utilities.

The **-w** option in early proposals was changed to **-l** to match historical practice.

The **-N** option was included because without it, a non-interactive application cannot reject previously applied patches. For example, if a user is piping the output of *diff* into the *patch* utility, and the user only wants to patch a file to a newer version non-interactively, the **-N** option is required.

Changes to the **-l** option description were proposed to allow matching across <newline> characters in addition to just <blank> characters. Since this is not historical practice, and since some ambiguities could result, it is suggested that future developments in this area utilize another option letter, such as **-L**.

The **-u** option of GNU *patch* has been added, along with support for unified context formats.

patch*Utilities*98945 **FUTURE DIRECTIONS**

98946 None.

98947 **SEE ALSO**98948 *diff, ed*98949 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)98950 **CHANGE HISTORY**

98951 First released in Issue 4.

98952 **Issue 5**

98953 The FUTURE DIRECTIONS section is added.

98954 **Issue 6**

98955 This utility is marked as part of the User Portability Utilities option.

98956 The description of the **-D** option and the steps in [Filename Determination](#) (on page 2997) are
98957 changed to match historical practice as defined in the IEEE P1003.2b draft standard.

98958 The normative text is reworded to avoid use of the term “must” for application requirements.

98959 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/34 is applied, clarifying the way that the
98960 *patch* utility performs **ifdef** selection for the **-D** option.98961 **Issue 7**98962 The *patch* utility is moved from the User Portability Utilities option to the Base. User Portability
98963 Utilities is now an option for interactive utilities.

98964 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

98965 SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the **-u** option.98966 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the
98967 *LC_MESSAGES* and *LC_CTYPE* environment variables and adding the *LC_COLLATE*
98968 environment variable.

Utilities

pathchk

NAME

pathchk — check pathnames

SYNOPSIS

pathchk [-p] [-P] *pathname*...

DESCRIPTION

The *pathchk* utility shall check that one or more pathnames are valid (that is, they could be used to access or create a file without causing syntax errors) and portable (that is, no filename truncation results). More extensive portability checks are provided by the **-p** and **-P** options.

By default, the *pathchk* utility shall check each component of each *pathname* operand based on the underlying file system. A diagnostic shall be written for each *pathname* operand that:

- Is longer than {PATH_MAX} bytes (see **Pathname Variable Values** in XBD Chapter 13 (on page 219), <limits.h>)
- Contains any component longer than {NAME_MAX} bytes in its containing directory
- Contains any component in a directory that is not searchable
- Contains any character in any component that is not valid in its containing directory

The format of the diagnostic message is not specified, but shall indicate the error detected and the corresponding *pathname* operand.

It shall not be considered an error if one or more components of a *pathname* operand do not exist as long as a file matching the pathname specified by the missing components could be created that does not violate any of the checks specified above.

OPTIONS

The *pathchk* utility shall conform to XBD Section 12.2 (on page 215).

The following option shall be supported:

- p** Instead of performing checks based on the underlying file system, write a diagnostic for each *pathname* operand that:
 - Is longer than {_POSIX_PATH_MAX} bytes (see **Minimum Values** in XBD Chapter 13 (on page 219), <limits.h>)
 - Contains any component longer than {_POSIX_NAME_MAX} bytes
 - Contains any character in any component that is not in the portable filename character set
- P** Write a diagnostic for each *pathname* operand that:
 - Contains a component whose first character is the <hyphen> character
 - Is empty

OPERANDS

The following operand shall be supported:

pathname A pathname to be checked.

STDIN

Not used.

pathchk

Utilities

99007 **INPUT FILES**

99008 None.

99009 **ENVIRONMENT VARIABLES**99010 The following environment variables shall affect the execution of *pathchk*:

99011 *LANG* Provide a default value for the internationalization variables that are unset or null.
 99012 (See XBD [Section 8.2](#) (on page 174) the precedence of internationalization variables
 99013 used to determine the values of locale categories.)

99014 *LC_ALL* If set to a non-empty string value, override the values of all the other
 99015 internationalization variables.

99016 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 99017 characters (for example, single-byte as opposed to multi-byte characters in
 99018 arguments).

99019 *LC_MESSAGES*

99020 Determine the locale that should be used to affect the format and contents of
 99021 diagnostic messages written to standard error.

99022 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

99023 **ASYNCHRONOUS EVENTS**

99024 Default.

99025 **STDOUT**

99026 Not used.

99027 **STDERR**

99028 The standard error shall be used only for diagnostic messages.

99029 **OUTPUT FILES**

99030 None.

99031 **EXTENDED DESCRIPTION**

99032 None.

99033 **EXIT STATUS**

99034 The following exit values shall be returned:

99035 0 All *pathname* operands passed all of the checks.

99036 >0 An error occurred.

99037 **CONSEQUENCES OF ERRORS**

99038 Default.

99039 **APPLICATION USAGE**

99040 The *test* utility can be used to determine whether a given *pathname* names an existing file; it
 99041 does not, however, give any indication of whether or not any component of the *pathname* was
 99042 truncated in a directory where the *_POSIX_NO_TRUNC* feature is not in effect. The *pathchk*
 99043 utility does not check for file existence; it performs checks to determine whether a *pathname*
 99044 does exist or could be created with no *pathname* component truncation.

99045 The *noclobber* option in the shell (see the *set* special built-in) can be used to atomically create a
 99046 file. As with all file creation semantics in the System Interfaces volume of POSIX.1-2008, it
 99047 guarantees atomic creation, but still depends on applications to agree on conventions and
 99048 cooperate on the use of files after they have been created.

99049 To verify that a *pathname* meets the requirements of filename portability, applications should

use both the `-p` and `-P` options together.

EXAMPLES

To verify that all pathnames in an imported data interchange archive are legitimate and unambiguous on the current system:

```
# This example assumes that no pathnames in the archive
# contain <newline> characters.
pax -f archive | sed -e 's/^[[:alnum:]]/\\&/g' | xargs pathchk --
if [ $? -eq 0 ]
then
    pax -r -f archive
else
    echo Investigate problems before importing files.
    exit 1
fi
```

To verify that all files in the current directory hierarchy could be moved to any system conforming to the System Interfaces volume of POSIX.1-2008 that also supports the *pax* utility:

```
find . -exec pathchk -p -P {} +
if [ $? -eq 0 ]
then
    pax -w -f ../archive .
else
    echo Portable archive cannot be created.
    exit 1
fi
```

To verify that a user-supplied pathname names a readable file and that the application can create a file extending the given path without truncation and without overwriting any existing file:

```
case $- in
    *C*)    reset="";;
    *)      reset="set +C"
           set -C;;
esac
test -r "$path" && pathchk "$path.out" &&
rm "$path.out" > "$path.out"
if [ $? -ne 0 ]; then
    printf "%s: %s not found or %s.out fails \
creation checks.\n" $0 "$path" "$path"
    $reset      # Reset the noclobber option in case a trap
               # on EXIT depends on it.
    exit 1
fi
$reset
PROCESSING < "$path" > "$path.out"
```

The following assumptions are made in this example:

1. **PROCESSING** represents the code that is used by the application to use **\$path** once it is verified that **\$path.out** works as intended.

pathchk*Utilities*

2. The state of the *noclobber* option is unknown when this code is invoked and should be set on exit to the state it was in when this code was invoked. (The **reset** variable is used in this example to restore the initial state.)

3. Note the usage of:

```
rm "$path.out" > "$path.out"
```

- a. The *pathchk* command has already verified, at this point, that **\$path.out** is not truncated.
- b. With the *noclobber* option set, the shell verifies that **\$path.out** does not already exist before invoking *rm*.
- c. If the shell succeeded in creating **\$path.out**, *rm* removes it so that the application can create the file again in the **PROCESSING** step.
- d. If the **PROCESSING** step wants the file to exist already when it is invoked, the:

```
rm "$path.out" > "$path.out"
```

should be replaced with:

```
> "$path.out"
```

which verifies that the file did not already exist, but leaves **\$path.out** in place for use by **PROCESSING**.

RATIONALE

The *pathchk* utility was new for the ISO POSIX-2:1993 standard. It, along with the *set -C(noclobber)* option added to the shell, replaces the *mktemp*, *validfnam*, and *create* utilities that appeared in early proposals. All of these utilities were attempts to solve several common problems:

- Verify the validity (for several different definitions of “valid”) of a pathname supplied by a user, generated by an application, or imported from an external source.
- Atomically create a file.
- Perform various string handling functions to generate a temporary filename.

The *create* utility, included in an early proposal, provided checking and atomic creation in a single invocation of the utility; these are orthogonal issues and need not be grouped into a single utility. Note that the *noclobber* option also provides a way of creating a lock for process synchronization; since it provides an atomic *create*, there is no race between a test for existence and the following creation if it did not exist.

Having a function like *tmpnam()* in the ISO C standard is important in many high-level languages. The shell programming language, however, has built-in string manipulation facilities, making it very easy to construct temporary filenames. The names needed obviously depend on the application, but are frequently of a form similar to:

```
$TMPDIR/application_abbreviation$$.suffix
```

In cases where there is likely to be contention for a given suffix, a simple shell **for** or **while** loop can be used with the shell *noclobber* option to create a file without risk of collisions, as long as applications trying to use the same filename name space are cooperating on the use of files after they have been created.

For historical purposes, **-p** does not check for the use of the <hyphen> character as the first character in a component of the pathname, or for an empty *pathname* operand.

99137 **FUTURE DIRECTIONS**

99138 None.

99139 **SEE ALSO**99140 [Section 2.7](#) (on page 2312), [set](#) (on page 2357), [test](#)99141 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215), [<limits.h>](#)99142 **CHANGE HISTORY**

99143 First released in Issue 4.

99144 **Issue 7**

99145 Austin Group Interpretations 1003.1-2001 #039, #040, and #094 are applied.

99146 SD5-XCU-ERN-121 is applied, updating the EXAMPLES section.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

pax*Utilities*99147 **NAME**99148 **pax** — portable archive interchange99149 **SYNOPSIS**99150 **pax** [-dv] [-c|-n] [-H|-L] [-o *options*] [-f *archive*] [-s *replstr*]...
99151 [*pattern*...]99152 **pax** -r[-c|-n] [-dikuv] [-H|-L] [-f *archive*] [-o *options*]... [-p *string*]...
99153 [-s *replstr*]... [*pattern*...]99154 **pax** -w [-dituvX] [-H|-L] [-b *blocksize*] [[-a] [-f *archive*]] [-o *options*]...
99155 [-s *replstr*]... [-x *format*] [*file*...]99156 **pax** -r -w [-diklntuvX] [-H|-L] [-o *options*]... [-p *string*]...
99157 [-s *replstr*]... [*file*...] *directory*99158 **DESCRIPTION**99159 The *pax* utility shall read, write, and write lists of the members of archive files and copy
99160 directory hierarchies. A variety of archive formats shall be supported; see the *-x format* option.99161 The action to be taken depends on the presence of the *-r* and *-w* options. The four combinations
99162 of *-r* and *-w* are referred to as the four modes of operation: **list**, **read**, **write**, and **copy** modes,
99163 corresponding respectively to the four forms shown in the SYNOPSIS section.99164 **list** In **list** mode (when neither *-r* nor *-w* are specified), *pax* shall write the names of
99165 the members of the archive file read from the standard input, with pathnames
99166 matching the specified patterns, to standard output. If a named file is of type
99167 directory, the file hierarchy rooted at that file shall be listed as well.99168 **read** In **read** mode (when *-r* is specified, but *-w* is not), *pax* shall extract the members of
99169 the archive file read from the standard input, with pathnames matching the
99170 specified patterns. If an extracted file is of type directory, the file hierarchy rooted
99171 at that file shall be extracted as well. The extracted files shall be created performing
99172 pathname resolution with the directory in which *pax* was invoked as the current
99173 working directory.99174 If an attempt is made to extract a directory when the directory already exists, this
99175 shall not be considered an error. If an attempt is made to extract a FIFO when the
99176 FIFO already exists, this shall not be considered an error.99177 The ownership, access, and modification times, and file mode of the restored files
99178 are discussed under the *-p* option.99179 **write** In **write** mode (when *-w* is specified, but *-r* is not), *pax* shall write the contents of
99180 the *file* operands to the standard output in an archive format. If no *file* operands are
99181 specified, a list of files to copy, one per line, shall be read from the standard input
99182 and each entry in this list shall be processed as if it had been a *file* operand on the
99183 command line. A file of type directory shall include all of the files in the file
99184 hierarchy rooted at the file.99185 **copy** In **copy** mode (when both *-r* and *-w* are specified), *pax* shall copy the *file* operands
99186 to the destination directory.99187 If no *file* operands are specified, a list of files to copy, one per line, shall be read
99188 from the standard input. A file of type directory shall include all of the files in the
99189 file hierarchy rooted at the file.99190 The effect of the **copy** shall be as if the copied files were written to a *pax* format
99191 archive file and then subsequently extracted, except that there may be hard links

between the original and the copied files. If the destination directory is a subdirectory of one of the files to be copied, the results are unspecified. If the destination directory is a file of a type not defined by the System Interfaces volume of POSIX.1-2008, the results are implementation-defined; otherwise, it shall be an error for the file named by the *directory* operand not to exist, not be writable by the user, or not be a file of type directory.

In **read** or **copy** modes, if intermediate directories are necessary to extract an archive member, *pax* shall perform actions equivalent to the *mkdir()* function defined in the System Interfaces volume of POSIX.1-2008, called with the following arguments:

- The intermediate directory used as the *path* argument
- The value of the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO as the *mode* argument

If any specified *pattern* or *file* operands are not matched by at least one file or archive member, *pax* shall write a diagnostic message to standard error for each one that did not match and exit with a non-zero exit status.

The archive formats described in the EXTENDED DESCRIPTION section shall be automatically detected on input. The default output archive format shall be implementation-defined.

A single archive can span multiple files. The *pax* utility shall determine, in an implementation-defined manner, what file to read or write as the next file.

If the selected archive format supports the specification of linked files, it shall be an error if these files cannot be linked when the archive is extracted. For archive formats that do not store file contents with each name that causes a hard link, if the file that contains the data is not extracted during this *pax* session, either the data shall be restored from the original file, or a diagnostic message shall be displayed with the name of a file that can be used to extract the data. In traversing directories, *pax* shall detect infinite loops; that is, entering a previously visited directory that is an ancestor of the last file visited. When it detects an infinite loop, *pax* shall write a diagnostic message to standard error and shall terminate.

OPTIONS

The *pax* utility shall conform to XBD Section 12.2 (on page 215), except that the order of presentation of the **-o**, **-p**, and **-s** options is significant.

The following options shall be supported:

- r** Read an archive file from standard input.
- w** Write files to the standard output in the specified archive format.
- a** Append files to the end of the archive. It is implementation-defined which devices on the system support appending. Additional file formats unspecified by this volume of POSIX.1-2008 may impose restrictions on appending.
- b *blocksize*** Block the output at a positive decimal integer number of bytes per write to the archive file. Devices and archive formats may impose restrictions on blocking. Blocking shall be automatically determined on input. Conforming applications shall not specify a *blocksize* value larger than 32 256. Default blocking when creating archives depends on the archive format. (See the **-x** option below.)
- c** Match all file or archive members except those specified by the *pattern* or *file* operands.

pax*Utilities*

99235	-d	Cause files of type directory being copied or archived or archive members of type directory being extracted or listed to match only the file or archive member itself and not the file hierarchy rooted at the file.
99236		
99237		
99238	-f archive	Specify the pathname of the input or output archive, overriding the default standard input (in list or read modes) or standard output (write mode).
99239		
99240	-H	If a symbolic link referencing a file of type directory is specified on the command line, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which <i>pax</i> can normally archive is specified on the command line, then <i>pax</i> shall archive the file referenced by the link, using the name of the link. The default behavior, when neither -H or -L are specified, shall be to archive the symbolic link itself.
99241		
99242		
99243		
99244		
99245		
99246		
99247	-i	Interactively rename files or archive members. For each archive member matching a <i>pattern</i> operand or file matching a <i>file</i> operand, a prompt shall be written to the file /dev/tty . The prompt shall contain the name of the file or archive member, but the format is otherwise unspecified. A line shall then be read from /dev/tty . If this line is blank, the file or archive member shall be skipped. If this line consists of a single period, the file or archive member shall be processed with no modification to its name. Otherwise, its name shall be replaced with the contents of the line. The <i>pax</i> utility shall immediately exit with a non-zero exit status if end-of-file is encountered when reading a response or if /dev/tty cannot be opened for reading and writing.
99248		
99249		
99250		
99251		
99252		
99253		
99254		
99255		
99256		
99257		The results of extracting a hard link to a file that has been renamed during extraction are unspecified.
99258		
99259	-k	Prevent the overwriting of existing files.
99260	-l	(The letter ell.) In copy mode, hard links shall be made between the source and destination file hierarchies whenever possible. If specified in conjunction with -H or -L , when a symbolic link is encountered, the hard link created in the destination file hierarchy shall be to the file referenced by the symbolic link. If specified when neither -H nor -L is specified, when a symbolic link is encountered, the implementation shall create a hard link to the symbolic link in the source file hierarchy or copy the symbolic link to the destination.
99261		
99262		
99263		
99264		
99265		
99266		
99267	-L	If a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which <i>pax</i> can normally archive is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file referenced by the link, using the name of the link. The default behavior, when neither -H or -L are specified, shall be to archive the symbolic link itself.
99268		
99269		
99270		
99271		
99272		
99273		
99274		
99275	-n	Select the first archive member that matches each <i>pattern</i> operand. No more than one archive member shall be matched for each pattern (although members of type directory shall still match the file hierarchy rooted at that file).
99276		
99277		
99278	-o options	Provide information to the implementation to modify the algorithm for extracting or writing files. The value of <i>options</i> shall consist of one or more <comma>-separated keywords of the form:
99279		
99280		
99281		<i>keyword</i> [[:]=value] [, <i>keyword</i> [[:]=value], ...]

Some keywords apply only to certain file formats, as indicated with each description. Use of keywords that are inapplicable to the file format being processed produces undefined results.

Keywords in the *options* argument shall be a string that would be a valid portable filename as described in XBD Section 3.276 (on page 77).

Note: Keywords are not expected to be filenames, merely to follow the same character composition rules as portable filenames.

Keywords can be preceded with white space. The *value* field shall consist of zero or more characters; within *value*, the application shall precede any literal <comma> with a <backslash>, which shall be ignored, but preserves the <comma> as part of *value*. A <comma> as the final character, or a <comma> followed solely by white space as the final characters, in *options* shall be ignored. Multiple **-o** options can be specified; if keywords given to these multiple **-o** options conflict, the keywords and values appearing later in command line sequence shall take precedence and the earlier shall be silently ignored. The following keyword values of *options* shall be supported for the file formats as indicated:

delete=pattern

(Applicable only to the **-x pax** format.) When used in **write** or **copy** mode, *pax* shall omit from extended header records that it produces any keywords matching the string pattern. When used in **read** or **list** mode, *pax* shall ignore any keywords matching the string pattern in the extended header records. In both cases, matching shall be performed using the pattern matching notation described in Section 2.13.1 (on page 2332) and Section 2.13.2 (on page 2332). For example:

-o delete=security.*

would suppress security-related information. See *pax Extended Header* (on page 3019) for extended header record keyword usage.

When multiple **-o delete=pattern** options are specified, the patterns shall be additive; all keywords matching the specified string patterns shall be omitted from extended header records that *pax* produces.

exthdr.name=string

(Applicable only to the **-x pax** format.) This keyword allows user control over the name that is written into the **ustar** header blocks for the extended header produced under the circumstances described in *pax Header Block* (on page 3019). The name shall be the contents of *string*, after the following character substitutions have been made:

<i>string</i> Includes:	Replaced by:
%d	The directory name of the file, equivalent to the result of the <i>dirname</i> utility on the translated pathname.
%f	The filename of the file, equivalent to the result of the <i>basename</i> utility on the translated pathname.
%p	The process ID of the <i>pax</i> process.
%%	A ' % ' character.

Any other ' % ' characters in *string* produce undefined results.

If no **-o exthdr.name=string** is specified, *pax* shall use the following default

pax*Utilities*

99328 value:
 99329 %d/PaxHeaders.%p/%f

99330 **globexthdr.name=string**

99331 (Applicable only to the **-x pax** format.) When used in **write** or **copy** mode
 99332 with the appropriate options, *pax* shall create global extended header records
 99333 with **ustar** header blocks that will be treated as regular files by previous
 99334 versions of *pax*. This keyword allows user control over the name that is
 99335 written into the **ustar** header blocks for global extended header records. The
 99336 name shall be the contents of *string*, after the following character substitutions
 99337 have been made:

<i>string</i> Includes:	Replaced by:
%n	An integer that represents the sequence number of the global extended header record in the archive, starting at 1.
%p	The process ID of the <i>pax</i> process.
%%	A '%' character.

99344 Any other '%' characters in *string* produce undefined results.

99345 If no **-o globexthdr.name=string** is specified, *pax* shall use the following
 99346 default value:

99347 \$TMPDIR/GlobalHead.%p.%n

99348 where *\$TMPDIR* represents the value of the *TMPDIR* environment variable. If
 99349 *TMPDIR* is not set, *pax* shall use **/tmp**.

99350 **invalid=action**

99351 (Applicable only to the **-x pax** format.) This keyword allows user control over
 99352 the action *pax* takes upon encountering values in an extended header record
 99353 that, in **read** or **copy** mode, are invalid in the destination hierarchy or, in **list**
 99354 mode, cannot be written in the codeset and current locale of the
 99355 implementation. The following are invalid values that shall be recognized by
 99356 *pax*:

- 99357 — In **read** or **copy** mode, a filename or link name that contains character
 99358 encodings invalid in the destination hierarchy. (For example, the name
 99359 may contain embedded NULs.)
- 99360 — In **read** or **copy** mode, a filename or link name that is longer than the
 99361 maximum allowed in the destination hierarchy (for either a pathname
 99362 component or the entire pathname).
- 99363 — In **list** mode, any character string value (filename, link name, user name,
 99364 and so on) that cannot be written in the codeset and current locale of the
 99365 implementation.

99366 The following mutually-exclusive values of the *action* argument are supported:

99367 **binary** In **write** mode, *pax* shall generate a **hdrcharset=BINARY**
 99368 extended header record for each file with a filename, link name,
 99369 group name, owner name, or any other field in an extended
 99370 header record that cannot be translated to the UTF-8 codeset,
 99371 allowing the archive to contain the files with unencoded
 99372 extended header record values. In **read** or **copy** mode, *pax* shall

99373		use the values specified in the header without translation,
99374		regardless of whether this may overwrite an existing file with a
99375		valid name. In list mode, <i>pax</i> shall behave identically to the
99376		bypass action.
99377	bypass	In read or copy mode, <i>pax</i> shall bypass the file, causing no
99378		change to the destination hierarchy. In list mode, <i>pax</i> shall write
99379		all requested valid values for the file, but its method for writing
99380		invalid values is unspecified.
99381	rename	In read or copy mode, <i>pax</i> shall act as if the -i option were in
99382		effect for each file with invalid filename or link name values,
99383		allowing the user to provide a replacement name interactively.
99384		In list mode, <i>pax</i> shall behave identically to the bypass action.
99385	UTF-8	When used in read , copy , or list mode and a filename, link
99386		name, owner name, or any other field in an extended header
99387		record cannot be translated from the pax UTF-8 codeset format
99388		to the codeset and current locale of the implementation, <i>pax</i> shall
99389		use the actual UTF-8 encoding for the name. If a hdrcharset
99390		extended header record is in effect for this file, the character set
99391		specified by that record shall be used instead of UTF-8. If a
99392		hdrcharset=BINARY extended header record is in effect for this
99393		file, no translation shall be performed.
99394	write	In read or copy mode, <i>pax</i> shall write the file, translating the
99395		name, regardless of whether this may overwrite an existing file
99396		with a valid name. In list mode, <i>pax</i> shall behave identically to
99397		the bypass action.
99398		If no -o invalid=option is specified, <i>pax</i> shall act as if -o invalid=bypass were
99399		specified. Any overwriting of existing files that may be allowed by the
99400		-o invalid= actions shall be subject to permission (-p) and modification time
99401		(-u) restrictions, and shall be suppressed if the -k option is also specified.
99402	linkdata	(Applicable only to the -x pax format.) In write mode, <i>pax</i> shall write the
99403		contents of a file to the archive even when that file is merely a hard link to a
99404		file whose contents have already been written to the archive.
99405		
99406	listopt=format	
99407		This keyword specifies the output format of the table of contents produced
99408		when the -v option is specified in list mode. See List Mode Format
99409		Specifications (on page 3014). To avoid ambiguity, the listopt=format shall be
99410		the only or final keyword=value pair in a -o option-argument; all characters
99411		in the remainder of the option-argument shall be considered part of the format
99412		string. When multiple -olistopt=format options are specified, the format
99413		strings shall be considered a single, concatenated string, evaluated in
99414		command line order.
99415	times	
99416		(Applicable only to the -x pax format.) When used in write or copy mode, <i>pax</i>
99417		shall include atime and mtime extended header records for each file. See pax
99418		Extended Header File Times (on page 3023).
99419		In addition to these keywords, if the -x pax format is specified, any of the

keywords and values defined in [pax Extended Header](#) (on page 3019), including implementation extensions, can be used in **-o** option-arguments, in either of two modes:

keyword=*value*

When used in **write** or **copy** mode, these keyword/value pairs shall be included at the beginning of the archive as **typeflag g** global extended header records. When used in **read** or **list** mode, these keyword/value pairs shall act as if they had been at the beginning of the archive as **typeflag g** global extended header records.

keyword:*value*

When used in **write** or **copy** mode, these keyword/value pairs shall be included as records at the beginning of a **typeflag x** extended header for each file. (This shall be equivalent to the **<equals-sign>** form except that it creates no **typeflag g** global extended header records.) When used in **read** or **list** mode, these keyword/value pairs shall act as if they were included as records at the end of each extended header; thus, they shall override any global or file-specific extended header record keywords of the same names. For example, in the command:

```
pax -r -o "  
gname:=mygroup,  
" <archive
```

the group name will be forced to a new value for all files read from the archive.

The precedence of **-o** keywords over various fields in the archive is described in [pax Extended Header Keyword Precedence](#) (on page 3022).

-p *string*

Specify one or more file characteristic options (privileges). The *string* option-argument shall be a string specifying file characteristics to be retained or discarded on extraction. The string shall consist of the specification characters **a**, **e**, **m**, **o**, and **p**. Other implementation-defined characters can be included. Multiple characteristics can be concatenated within the same string and multiple **-p** options can be specified. The meaning of the specification characters are as follows:

- a** Do not preserve file access times.
- e** Preserve the user ID, group ID, file mode bits (see [XBD Section 3.169](#), on page 60), access time, modification time, and any other implementation-defined file characteristics.
- m** Do not preserve file modification times.
- o** Preserve the user ID and group ID.
- p** Preserve the file mode bits. Other implementation-defined file mode attributes may be preserved.

In the preceding list, “preserve” indicates that an attribute stored in the archive shall be given to the extracted file, subject to the permissions of the invoking process. The access and modification times of the file shall be preserved unless otherwise specified with the **-p** option or not stored in the archive. All attributes that are not preserved shall be determined as part of the normal file creation action (see [Section 1.1.1.4](#), on page 2280).

Utilities

pax

99465		If neither the <i>e</i> nor the <i>o</i> specification character is specified, or the user ID and
99466		group ID are not preserved for any reason, <i>pax</i> shall not set the <i>S_ISUID</i> and
99467		<i>S_ISGID</i> bits of the file mode.
99468		If the preservation of any of these items fails for any reason, <i>pax</i> shall write a
99469		diagnostic message to standard error. Failure to preserve these items shall affect
99470		the final exit status, but shall not cause the extracted file to be deleted.
99471		If file characteristic letters in any of the <i>string</i> option-arguments are duplicated or
99472		conflict with each other, the ones given last shall take precedence. For example, if
99473		-p <i>eme</i> is specified, file modification times are preserved.
99474	-s replstr	Modify file or archive member names named by <i>pattern</i> or <i>file</i> operands according
99475		to the substitution expression <i>replstr</i> , using the syntax of the <i>ed</i> utility. The concepts
99476		of “address” and “line” are meaningless in the context of the <i>pax</i> utility, and shall
99477		not be supplied. The format shall be:
99478		<i>-s /old/new/[gp]</i>
99479		where as in <i>ed</i> , <i>old</i> is a basic regular expression and <i>new</i> can contain an
99480		<ampersand>, ‘\n’ (where <i>n</i> is a digit) back-references, or subexpression
99481		matching. The <i>old</i> string shall also be permitted to contain <newline> characters.
99482		Any non-null character can be used as a delimiter (‘/’ shown here). Multiple -s
99483		expressions can be specified; the expressions shall be applied in the order
99484		specified, terminating with the first successful substitution. The optional trailing
99485		‘g’ is as defined in the <i>ed</i> utility. The optional trailing ‘p’ shall cause successful
99486		substitutions to be written to standard error. File or archive member names that
99487		substitute to the empty string shall be ignored when reading and writing archives.
99488	-t	When reading files from the file system, and if the user has the permissions
99489		required by <i>utime()</i> to do so, set the access time of each file read to the access time
99490		that it had before being read by <i>pax</i> .
99491	-u	Ignore files that are older (having a less recent file modification time) than a pre-
99492		existing file or archive member with the same name. In read mode, an archive
99493		member with the same name as a file in the file system shall be extracted if the
99494		archive member is newer than the file. In write mode, an archive file member with
99495		the same name as a file in the file system shall be superseded if the file is newer
99496		than the archive member. If -a is also specified, this is accomplished by appending
99497		to the archive; otherwise, it is unspecified whether this is accomplished by actual
99498		replacement in the archive or by appending to the archive. In copy mode, the file
99499		in the destination hierarchy shall be replaced by the file in the source hierarchy or
99500		by a link to the file in the source hierarchy if the file in the source hierarchy is
99501		newer.
99502	-v	In list mode, produce a verbose table of contents (see the STDOUT section).
99503		Otherwise, write archive member pathnames to standard error (see the STDERR
99504		section).
99505	-x format	Specify the output archive format. The <i>pax</i> utility shall support the following
99506		formats:
99507	cpio	The cpio interchange format; see the EXTENDED DESCRIPTION
99508		section. The default <i>blocksize</i> for this format for character special
99509		archive files shall be 5120. Implementations shall support all
99510		<i>blocksize</i> values less than or equal to 32 256 that are multiples of 512.

pax*Utilities*

99511	pax	The pax interchange format; see the EXTENDED DESCRIPTION
99512		section. The default <i>blocksize</i> for this format for character special
99513		archive files shall be 5120. Implementations shall support all
99514		<i>blocksize</i> values less than or equal to 32 256 that are multiples of 512.
99515	ustar	The tar interchange format; see the EXTENDED DESCRIPTION
99516		section. The default <i>blocksize</i> for this format for character special
99517		archive files shall be 10 240. Implementations shall support all
99518		<i>blocksize</i> values less than or equal to 32 256 that are multiples of 512.
99519		Implementation-defined formats shall specify a default block size as well as any
99520		other block sizes supported for character special archive files.
99521		Any attempt to append to an archive file in a format different from the existing
99522		archive format shall cause <i>pax</i> to exit immediately with a non-zero exit status.
99523	-X	When traversing the file hierarchy specified by a pathname, <i>pax</i> shall not descend
99524		into directories that have a different device ID (<i>st_dev</i> ; see the System Interfaces
99525		volume of POSIX.1-2008, <i>stat()</i>).
99526		Specifying more than one of the mutually-exclusive options -H and -L shall not be considered
99527		an error and the last option specified shall determine the behavior of the utility.
99528		The options that operate on the names of files or archive members (-c , -i , -n , -s , -u , and -v)
99529		shall interact as follows. In read mode, the archive members shall be selected based on the user-
99530		specified <i>pattern</i> operands as modified by the -c , -n , and -u options. Then, any -s and -i
99531		options shall modify, in that order, the names of the selected files. The -v option shall write
99532		names resulting from these modifications.
99533		In write mode, the files shall be selected based on the user-specified pathnames as modified by
99534		the -n and -u options. Then, any -s and -i options shall modify, in that order, the names of
99535		these selected files. The -v option shall write names resulting from these modifications.
99536		If both the -u and -n options are specified, <i>pax</i> shall not consider a file selected unless it is
99537		newer than the file to which it is compared.
99538		List Mode Format Specifications
99539		In list mode with the -o listopt=format option, the <i>format</i> argument shall be applied for each
99540		selected file. The <i>pax</i> utility shall append a <newline> to the listopt output for each selected file.
99541		The <i>format</i> argument shall be used as the <i>format</i> string described in XBD Chapter 5 (on page 121),
99542		with the exceptions 1. through 5. defined in the EXTENDED DESCRIPTION section of <i>printf</i> ,
99543		plus the following exceptions:
99544		6. The sequence (<i>keyword</i>) can occur before a format conversion specifier. The conversion
99545		argument is defined by the value of <i>keyword</i> . The implementation shall support the
99546		following keywords:
99547		— Any of the Field Name entries in Table 4-14 (on page 3024) and Table 4-16 (on page
99548		3027). The implementation may support the <i>cpio</i> keywords without the leading <i>c_</i> in
99549		addition to the form required by Table 4-16 (on page 3027).
99550		— Any keyword defined for the extended header in <i>pax Extended Header</i> (on page
99551		3019).
99552		— Any keyword provided as an implementation-defined extension within the extended
99553		header defined in <i>pax Extended Header</i> (on page 3019).
99554		For example, the sequence "%(charset)s" is the string value of the name of the character

- 99555 set in the extended header.
- 99556 The result of the keyword conversion argument shall be the value from the applicable
99557 header field or extended header, without any trailing NULs.
- 99558 All keyword values used as conversion arguments shall be translated from the UTF-8
99559 encoding (or alternative encoding specified by any **hdrcharset** extended header record) to
99560 the character set appropriate for the local file system, user database, and so on, as
99561 applicable.
- 99562 7. An additional conversion specifier character, **T**, shall be used to specify time formats. The **T**
99563 conversion specifier character can be preceded by the sequence (*keyword=subformat*), where
99564 *subformat* is a date format as defined by *date* operands. The default *keyword* shall be **mtime**
99565 and the default subformat shall be:
- 99566 %b %e %H:%M %Y
- 99567 8. An additional conversion specifier character, **M**, shall be used to specify the file mode string
99568 as defined in *ls* Standard Output. If (*keyword*) is omitted, the **mode** keyword shall be used.
99569 For example, % . 1M writes the single character corresponding to the <entry type> field of the
99570 *ls -l* command.
- 99571 9. An additional conversion specifier character, **D**, shall be used to specify the device for block
99572 or special files, if applicable, in an implementation-defined format. If not applicable, and
99573 (*keyword*) is specified, then this conversion shall be equivalent to % (*keyword*)u. If not
99574 applicable, and (*keyword*) is omitted, then this conversion shall be equivalent to <space>.
- 99575 10. An additional conversion specifier character, **F**, shall be used to specify a pathname. The **F**
99576 conversion character can be preceded by a sequence of <comma>-separated keywords:
- 99577 (*keyword* [, *keyword*] ...)
- 99578 The values for all the keywords that are non-null shall be concatenated together, each
99579 separated by a ' / '. The default shall be (**path**) if the keyword **path** is defined; otherwise,
99580 the default shall be (**prefix,name**).
- 99581 11. An additional conversion specifier character, **L**, shall be used to specify a symbolic link
99582 expansion. If the current file is a symbolic link, then %L shall expand to:
- 99583 "%s -> %s", <value of keyword>, <contents of link>
- 99584 Otherwise, the %L conversion specification shall be the equivalent of %F.

99585 OPERANDS

99586 The following operands shall be supported:

- 99587 *directory* The destination directory pathname for **copy** mode.
- 99588 *file* A pathname of a file to be copied or archived.
- 99589 *pattern* A pattern matching one or more pathnames of archive members. A pattern must
99590 be given in the name-generating notation of the pattern matching notation in
99591 [Section 2.13](#) (on page 2332), including the filename expansion rules in [Section](#)
99592 [2.13.3](#) (on page 2333). The default, if no *pattern* is specified, is to select all members
99593 in the archive.

99594 STDIN

99595 In **write** mode, the standard input shall be used only if no *file* operands are specified. It shall be a
99596 text file containing a list of pathnames, one per line, without leading or trailing <blank>
99597 characters.

pax*Utilities*

99598 In **list** and **read** modes, if **-f** is not specified, the standard input shall be an archive file.

99599 Otherwise, the standard input shall not be used.

INPUT FILES

99601 The input file named by the *archive* option-argument, or standard input when the archive is read
99602 from there, shall be a file formatted according to one of the specifications in the EXTENDED
99603 DESCRIPTION section or some other implementation-defined format.

99604 The file **/dev/tty** shall be used to write prompts and read responses.

ENVIRONMENT VARIABLES

99605 The following environment variables shall affect the execution of *pax*:

99607 **LANG** Provide a default value for the internationalization variables that are unset or null.
99608 (See XBD [Section 8.2](#) (on page 174) the precedence of internationalization variables
99609 used to determine the values of locale categories.)

99610 **LC_ALL** If set to a non-empty string value, override the values of all the other
99611 internationalization variables.

99612 **LC_COLLATE**

99613 Determine the locale for the behavior of ranges, equivalence classes, and multi-
99614 character collating elements used in the pattern matching expressions for the
99615 *pattern* operand, the basic regular expression for the **-s** option, and the extended
99616 regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES*
99617 category.

99618 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
99619 characters (for example, single-byte as opposed to multi-byte characters in
99620 arguments and input files), the behavior of character classes used in the extended
99621 regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES*
99622 category, and pattern matching.

99623 **LC_MESSAGES**

99624 Determine the locale used to process affirmative responses, and the locale used to
99625 affect the format and contents of diagnostic messages and prompts written to
99626 standard error.

99627 **LC_TIME** Determine the format and contents of date and time strings when the **-v** option is
99628 specified.

99629 **XSIX** **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

99630 **TMPDIR** Determine the pathname that provides part of the default global extended header
99631 record file, as described for the **-o globexthdr=** keyword in the **OPTIONS** section.

99632 **TZ** Determine the timezone used to calculate date and time strings when the **-v** option
99633 is specified. If *TZ* is unset or null, an unspecified default timezone shall be used.

ASYNCHRONOUS EVENTS

99635 Default.

STDOUT

99637 In **write** mode, if **-f** is not specified, the standard output shall be the archive formatted
99638 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other
99639 implementation-defined format (see **-x format**).

99640 In **list** mode, when the **-olistopt=format** has been specified, the selected archive members shall
99641 be written to standard output using the format described under [List Mode Format Specifications](#)

(on page 3014). In **list** mode without the **-olistopt=format** option, the table of contents of the selected archive members shall be written to standard output using the following format:

```
"%s\n", <pathname>
```

If the **-v** option is specified in **list** mode, the table of contents of the selected archive members shall be written to standard output using the following formats.

For pathnames representing hard links to previous members of the archive:

```
"%sΔ==Δ%s\n", <ls -l listing>, <linkname>
```

For all other pathnames:

```
"%s\n", <ls -l listing>
```

where *<ls -l listing>* shall be the format specified by the *ls* utility with the **-l** option. When writing pathnames in this format, it is unspecified what is written for fields for which the underlying archive format does not have the correct information, although the correct number of *<blank>*-separated fields shall be written.

In **list** mode, standard output shall not be buffered more than a line at a time.

STDERR

If **-v** is specified in **read**, **write**, or **copy** modes, *pax* shall write the pathnames it processes to the standard error output using the following format:

```
"%s\n", <pathname>
```

These pathnames shall be written as soon as processing is begun on the file or archive member, and shall be flushed to standard error. The trailing *<newline>*, which shall not be buffered, is written when the file has been read or written.

If the **-s** option is specified, and the replacement string has a trailing 'p', substitutions shall be written to standard error in the following format:

```
"%sΔ>>Δ%s\n", <original pathname>, <new pathname>
```

In all operating modes of *pax*, optional messages of unspecified format concerning the input archive format and volume number, the number of files, blocks, volumes, and media parts as well as other diagnostic messages may be written to standard error.

In all formats, for both standard output and standard error, it is unspecified how non-printable characters in pathnames or link names are written.

When using the **-xpax** archive format, if a filename, link name, group name, owner name, or any other field in an extended header record cannot be translated between the codeset in use for that extended header record and the character set of the current locale, *pax* shall write a diagnostic message to standard error, shall process the file as described for the **-o invalid=** option, and then shall continue processing with the next file.

OUTPUT FILES

In **read** mode, the extracted output files shall be of the archived file type. In **copy** mode, the copied output files shall be the type of the file being copied. In either mode, existing files in the destination hierarchy shall be overwritten only when all permission (**-p**), modification time (**-u**), and invalid-value (**-o invalid=**) tests allow it.

In **write** mode, the output file named by the **-f** option-argument shall be a file formatted according to one of the specifications in the EXTENDED DESCRIPTION section, or some other implementation-defined format.

99684 EXTENDED DESCRIPTION

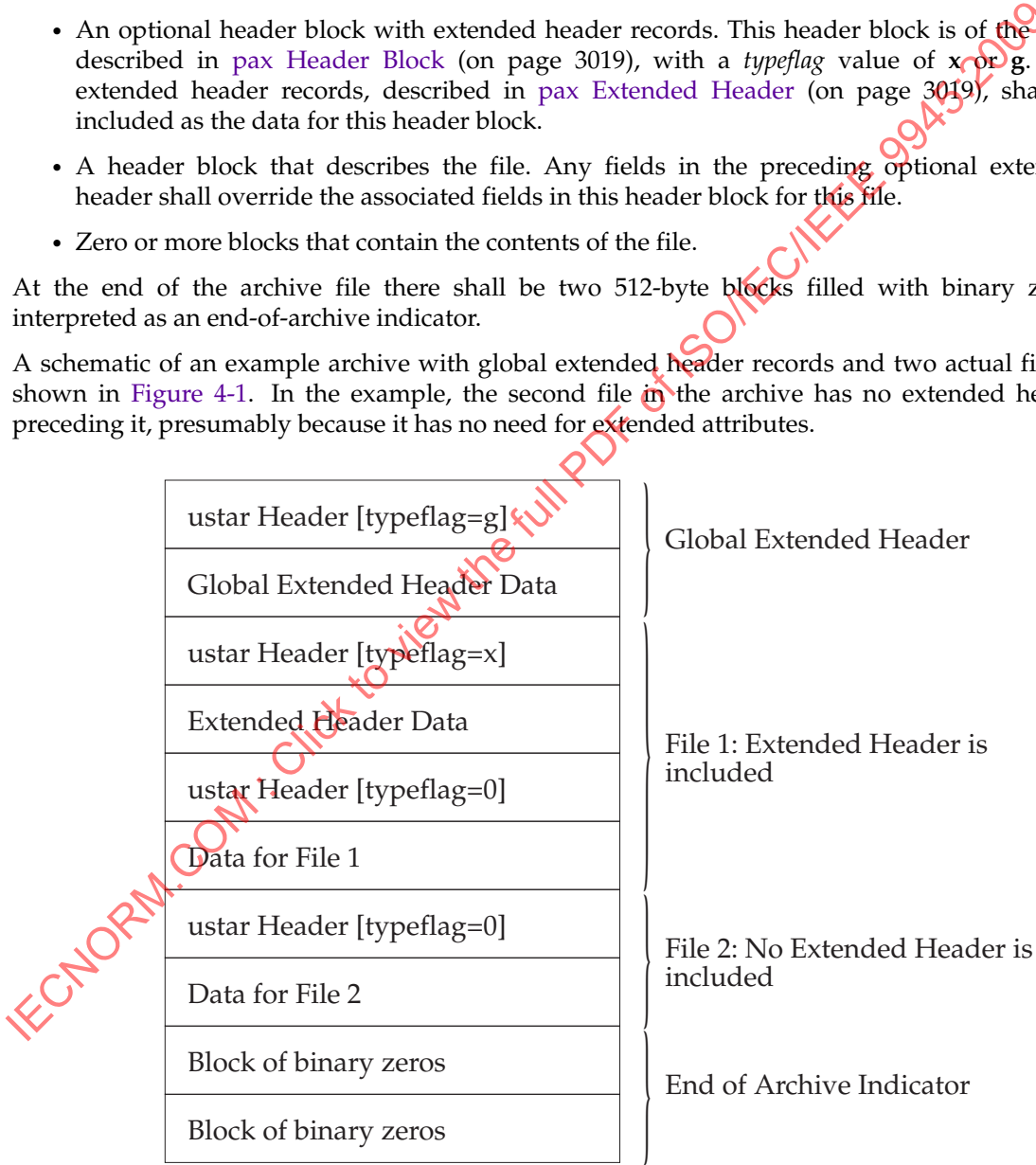
99685 **pax Interchange Format**

99686 A *pax* archive tape or file produced in the **-xpax** format shall contain a series of blocks. The
99687 physical layout of the archive shall be identical to the **ustar** format described in **ustar**
99688 **Interchange Format** (on page 3023). Each file archived shall be represented by the following
99689 sequence:

- 99690 • An optional header block with extended header records. This header block is of the form
99691 described in **pax Header Block** (on page 3019), with a *typeflag* value of **x** or **g**. The
99692 extended header records, described in **pax Extended Header** (on page 3019), shall be
99693 included as the data for this header block.
- 99694 • A header block that describes the file. Any fields in the preceding optional extended
99695 header shall override the associated fields in this header block for this file.
- 99696 • Zero or more blocks that contain the contents of the file.

99697 At the end of the archive file there shall be two 512-byte blocks filled with binary zeros,
99698 interpreted as an end-of-archive indicator.

99699 A schematic of an example archive with global extended header records and two actual files is
99700 shown in **Figure 4-1**. In the example, the second file in the archive has no extended header
99701 preceding it, presumably because it has no need for extended attributes.



99702 **Figure 4-1** pax Format Archive Example

pax Header Block

The **pax** header block shall be identical to the **ustar** header block described in **ustar Interchange Format** (on page 3023), except that two additional *typeflag* values are defined:

- x Represents extended header records for the following file in the archive (which shall have its own **ustar** header block). The format of these extended header records shall be as described in **pax Extended Header**.
- g Represents global extended header records for the following files in the archive. The format of these extended header records shall be as described in **pax Extended Header**. Each value shall affect all subsequent files that do not override that value in their own extended header record and until another global extended header record is reached that provides another value for the same field. The *typeflag* **g** global headers should not be used with interchange media that could suffer partial data loss in transporting the archive.

For both of these types, the *size* field shall be the size of the extended header records in octets. The other fields in the header block are not meaningful to this version of the *pax* utility. However, if this archive is read by a *pax* utility conforming to the ISO POSIX-2: 1993 standard, the header block fields are used to create a regular file that contains the extended header records as data. Therefore, header block field values should be selected to provide reasonable file access to this regular file.

A further difference from the **ustar** header block is that data blocks for files of *typeflag* 1 (the digit one) (hard link) may be included, which means that the size field may be greater than zero. Archives created by *pax* **-o linkdata** shall include these data blocks with the hard links.

pax Extended Header

A **pax** extended header contains values that are inappropriate for the **ustar** header block because of limitations in that format: fields requiring a character encoding other than that described in the ISO/IEC 646: 1991 standard, fields representing file attributes not described in the **ustar** header, and fields whose format or length do not fit the requirements of the **ustar** header. The values in an extended header add attributes to the following file (or files; see the description of the *typeflag* **g** header block) or override values in the following header block(s), as indicated in the following list of keywords.

An extended header shall consist of one or more records, each constructed as follows:

```
"%d %s=%s\n", <length>, <keyword>, <value>
```

The extended header records shall be encoded according to the ISO/IEC 10646-1: 2000 standard UTF-8 encoding. The *<length>* field, *<blank>*, *<equals-sign>*, and *<newline>* shown shall be limited to the portable character set, as encoded in UTF-8. The *<keyword>* fields can be any UTF-8 characters. The *<length>* field shall be the decimal length of the extended header record in octets, including the trailing *<newline>*. If there is a **hdrcharset** extended header in effect for a file, the *value* field for any **gname**, **linkpath**, **path**, and **uname** extended header records shall be encoded using the character set specified by the **hdrcharset** extended header record; otherwise, the *value* field shall be encoded using UTF-8. The *value* field for all other keywords specified by POSIX.1-2008 shall be encoded using UTF-8.

The *<keyword>* field shall be one of the entries from the following list or a keyword provided as an implementation extension. Keywords consisting entirely of lowercase letters, digits, and periods are reserved for future standardization. A keyword shall not include an *<equals-sign>*. (In the following list, the notations "file(s)" or "block(s)" is used to acknowledge that a keyword affects the following single file after a *typeflag* **x** extended header, but possibly multiple files after *typeflag* **g**. Any requirements in the list for *pax* to include a record when in **write** or **copy** mode

pax*Utilities*

shall apply only when such a record has not already been provided through the use of the **-o** option. When used in **copy** mode, *pax* shall behave as if an archive had been created with applicable extended header records and then extracted.)

atime The file access time for the following file(s), equivalent to the value of the *st_atime* member of the **stat** structure for a file, as described by the *stat()* function. The access time shall be restored if the process has appropriate privileges required to do so. The format of the *<value>* shall be as described in **pax Extended Header File Times** (on page 3023).

charset The name of the character set used to encode the data in the following file(s). The entries in the following table are defined to refer to known standards; additional names may be agreed on between the originator and recipient.

<i><value></i>	Formal Standard
ISO-IRΔ646Δ1990	ISO/IEC 646: 1990
ISO-IRΔ8859Δ1Δ1998	ISO/IEC 8859-1: 1998
ISO-IRΔ8859Δ2Δ1999	ISO/IEC 8859-2: 1999
ISO-IRΔ8859Δ3Δ1999	ISO/IEC 8859-3: 1999
ISO-IRΔ8859Δ4Δ1998	ISO/IEC 8859-4: 1998
ISO-IRΔ8859Δ5Δ1999	ISO/IEC 8859-5: 1999
ISO-IRΔ8859Δ6Δ1999	ISO/IEC 8859-6: 1999
ISO-IRΔ8859Δ7Δ1987	ISO/IEC 8859-7: 1987
ISO-IRΔ8859Δ8Δ1999	ISO/IEC 8859-8: 1999
ISO-IRΔ8859Δ9Δ1999	ISO/IEC 8859-9: 1999
ISO-IRΔ8859Δ10Δ1998	ISO/IEC 8859-10: 1998
ISO-IRΔ8859Δ13Δ1998	ISO/IEC 8859-13: 1998
ISO-IRΔ8859Δ14Δ1998	ISO/IEC 8859-14: 1998
ISO-IRΔ8859Δ15Δ1999	ISO/IEC 8859-15: 1999
ISO-IRΔ10646Δ2000	ISO/IEC 10646: 2000
ISO-IRΔ10646Δ2000ΔUTF-8	ISO/IEC 10646, UTF-8 encoding
BINARY	None.

The encoding is included in an extended header for information only; when *pax* is used as described in POSIX.1-2008, it shall not translate the file data into any other encoding. The **BINARY** entry indicates unencoded binary data.

When used in **write** or **copy** mode, it is implementation-defined whether *pax* includes a **charset** extended header record for a file.

comment A series of characters used as a comment. All characters in the *<value>* field shall be ignored by *pax*.

gid The group ID of the group that owns the file, expressed as a decimal number using digits from the ISO/IEC 646: 1991 standard. This record shall override the *gid* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *gid* extended header record for each file whose group ID is greater than 2 097 151 (octal 7 777 777).

gname The group of the file(s), formatted as a group name in the group database. This record shall override the *gid* and *gname* fields in the following header block(s), and any *gid* extended header record. When used in **read**, **copy**, or **list** mode, *pax* shall translate the name from the encoding in the header record to the character set appropriate for the group database on the receiving system. If any of the characters cannot be translated, and if neither the **-oinvalid=UTF-8** option nor the

–**oinvalid=binary** option is specified, the results are implementation-defined. When used in **write** or **copy** mode, *pax* shall include a **gname** extended header record for each file whose group name cannot be represented entirely with the letters and digits of the portable character set.

hdrcharset The name of the character set used to encode the value field of the **gname**, **linkpath**, **path**, and **uname** *pax* extended header records. The entries in the following table are defined to refer to known standards; additional names may be agreed between the originator and the recipient.

<value>	Formal Standard
ISO-IRΔ10646Δ2000ΔUTF-8	ISO/IEC 10646, UTF-8 encoding
BINARY	None.

If no **hdrcharset** extended header record is specified, the default character set used to encode all values in extended header records shall be the ISO/IEC 10646-1: 2000 standard UTF-8 encoding.

The **BINARY** entry indicates that all values recorded in extended headers for affected files are unencoded binary data from the underlying system.

linkpath The pathname of a link being created to another file, of any type, previously archived. This record shall override the *linkname* field in the following **ustar** header block(s). The following **ustar** header block shall determine the type of link created. If *typeflag* of the following header block is 1, it shall be a hard link. If *typeflag* is 2, it shall be a symbolic link and the **linkpath** value shall be the contents of the symbolic link. The *pax* utility shall translate the name of the link (contents of the symbolic link) from the encoding in the header to the character set appropriate for the local file system. When used in **write** or **copy** mode, *pax* shall include a **linkpath** extended header record for each link whose pathname cannot be represented entirely with the members of the portable character set other than NUL.

mtime The file modification time of the following file(s), equivalent to the value of the *st_mtime* member of the **stat** structure for a file, as described in the *stat()* function. This record shall override the *mtime* field in the following header block(s). The modification time shall be restored if the process has appropriate privileges required to do so. The format of the <value> shall be as described in [pax Extended Header File Times](#) (on page 3023).

path The pathname of the following file(s). This record shall override the *name* and *prefix* fields in the following header block(s). The *pax* utility shall translate the pathname of the file from the encoding in the header to the character set appropriate for the local file system.

When used in **write** or **copy** mode, *pax* shall include a *path* extended header record for each file whose pathname cannot be represented entirely with the members of the portable character set other than NUL.

realtime.any The keywords prefixed by “realtime.” are reserved for future standardization.

security.any The keywords prefixed by “security.” are reserved for future standardization.

size The size of the file in octets, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the *size* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *size* extended header record for each file with a size value greater than 8 589 934 591

- 99842 (octal 77 777 777 777).
- 99843 **uid** The user ID of the file owner, expressed as a decimal number using digits from the
 99844 ISO/IEC 646:1991 standard. This record shall override the *uid* field in the
 99845 following header block(s). When used in **write** or **copy** mode, *pax* shall include a
 99846 *uid* extended header record for each file whose owner ID is greater than 2097151
 99847 (octal 7777777).
- 99848 **uname** The owner of the following file(s), formatted as a user name in the user database.
 99849 This record shall override the *uid* and *uname* fields in the following header block(s),
 99850 and any *uid* extended header record. When used in **read**, **copy**, or **list** mode, *pax*
 99851 shall translate the name from the encoding in the header record to the character set
 99852 appropriate for the user database on the receiving system. If any of the characters
 99853 cannot be translated, and if neither the **-oinvalid=UTF-8** option nor the
 99854 **-oinvalid=binary** option is specified, the results are implementation-defined.
 99855 When used in **write** or **copy** mode, *pax* shall include a **uname** extended header
 99856 record for each file whose user name cannot be represented entirely with the letters
 99857 and digits of the portable character set.
- 99858 If the *<value>* field is zero length, it shall delete any header block field, previously entered
 99859 extended header value, or global extended header value of the same name.
- 99860 If a keyword in an extended header record (or in a **-o** option-argument) overrides or deletes a
 99861 corresponding field in the **ustar** header block, *pax* shall ignore the contents of that header block
 99862 field.
- 99863 Unlike the **ustar** header block fields, NULs shall not delimit *<value>*s; all characters within the
 99864 *<value>* field shall be considered data for the field. None of the length limitations of the **ustar**
 99865 header block fields in Table 4-14 (on page 3024) shall apply to the extended header records.
- 99866 **pax Extended Header Keyword Precedence**
- 99867 This section describes the precedence in which the various header records and fields and
 99868 command line options are selected to apply to a file in the archive. When *pax* is used in **read** or
 99869 **list** modes, it shall determine a file attribute in the following sequence:
- 99870 1. If **-odelete=keyword-prefix** is used, the affected attributes shall be determined from step
 99871 7., if applicable, or ignored otherwise.
 - 99872 2. If **-okeyword:=** is used, the affected attributes shall be ignored.
 - 99873 3. If **-okeyword:=value** is used, the affected attribute shall be assigned the value.
 - 99874 4. If there is a *typeflag* **x** extended header record, the affected attribute shall be assigned the
 99875 *<value>*. When extended header records conflict, the last one given in the header shall
 99876 take precedence.
 - 99877 5. If **-okeyword=value** is used, the affected attribute shall be assigned the value.
 - 99878 6. If there is a *typeflag* **g** global extended header record, the affected attribute shall be
 99879 assigned the *<value>*. When global extended header records conflict, the last one given in
 99880 the global header shall take precedence.
 - 99881 7. Otherwise, the attribute shall be determined from the **ustar** header block.

pax Extended Header File Times

The *pax* utility shall write an **mtime** record for each file in **write** or **copy** modes if the file's modification time cannot be represented exactly in the **ustar** header logical record described in **ustar Interchange Format**. This can occur if the time is out of **ustar** range, or if the file system of the underlying implementation supports non-integer time granularities and the time is not an integer. All of these time records shall be formatted as a decimal representation of the time in seconds since the Epoch. If a <period> (' . ') decimal point character is present, the digits to the right of the point shall represent the units of a subsecond timing granularity, where the first digit is tenths of a second and each subsequent digit is a tenth of the previous digit. In **read** or **copy** mode, the *pax* utility shall truncate the time of a file to the greatest value that is not greater than the input header file time. In **write** or **copy** mode, the *pax* utility shall output a time exactly if it can be represented exactly as a decimal number, and otherwise shall generate only enough digits so that the same time shall be recovered if the file is extracted on a system whose underlying implementation supports the same time granularity.

ustar Interchange Format

A **ustar** archive tape or file shall contain a series of logical records. Each logical record shall be a fixed-size logical record of 512 octets (see below). Although this format may be thought of as being stored on 9-track industry-standard 12.7 mm (0.5 in) magnetic tape, other types of transportable media are not excluded. Each file archived shall be represented by a header logical record that describes the file, followed by zero or more logical records that give the contents of the file. At the end of the archive file there shall be two 512-octet logical records filled with binary zeros, interpreted as an end-of-archive indicator.

The logical records may be grouped for physical I/O operations, as described under the **-bblocksize** and **-x ustar** options. Each group of logical records may be written with a single operation equivalent to the *write()* function. On magnetic tape, the result of this write shall be a single tape physical block. The last physical block shall always be the full size, so logical records after the two zero logical records may contain undefined data.

The header logical record shall be structured as shown in the following table. All lengths and offsets are in decimal.

Table 4-14 ustar Header Block

Field Name	Octet Offset	Length (in Octets)
<i>name</i>	0	100
<i>mode</i>	100	8
<i>uid</i>	108	8
<i>gid</i>	116	8
<i>size</i>	124	12
<i>mtime</i>	136	12
<i>chksum</i>	148	8
<i>typeflag</i>	156	1
<i>linkname</i>	157	100
<i>magic</i>	257	6
<i>version</i>	263	2
<i>uname</i>	265	32
<i>gname</i>	297	32
<i>devmajor</i>	329	8
<i>devminor</i>	337	8
<i>prefix</i>	345	155

All characters in the header logical record shall be represented in the coded character set of the ISO/IEC 646:1991 standard. For maximum portability between implementations, names should be selected from characters represented by the portable filename character set as octets with the most significant bit zero. If an implementation supports the use of characters outside of <slash> and the portable filename character set in names for files, users, and groups, one or more implementation-defined encodings of these characters shall be provided for interchange purposes.

However, the *pax* utility shall never create filenames on the local system that cannot be accessed via the procedures described in POSIX.1-2008. If a filename is found on the medium that would create an invalid filename, it is implementation-defined whether the data from the file is stored on the file hierarchy and under what name it is stored. The *pax* utility may choose to ignore these files as long as it produces an error indicating that the file is being ignored.

Each field within the header logical record is contiguous; that is, there is no padding used. Each character on the archive medium shall be stored contiguously.

The fields *magic*, *uname*, and *gname* are character strings each terminated by a NUL character. The fields *name*, *linkname*, and *prefix* are NUL-terminated character strings except when all characters in the array contain non-NUL characters including the last character. The *version* field is two octets containing the characters "00" (zero-zero). The *typeflag* contains a single character. All other fields are leading zero-filled octal numbers using digits from the ISO/IEC 646:1991 standard IRV. Each numeric field is terminated by one or more <space> or NUL characters.

The *name* and the *prefix* fields shall produce the pathname of the file. A new pathname shall be formed, if *prefix* is not an empty string (its first character is not NUL), by concatenating *prefix* (up to the first NUL character), a <slash> character, and *name*; otherwise, *name* is used alone. In either case, *name* is terminated at the first NUL character. If *prefix* begins with a NUL character, it shall be ignored. In this manner, pathnames of at most 256 characters can be supported. If a pathname does not fit in the space provided, *pax* shall notify the user of the error, and shall not store any part of the file—header or data—on the medium.

The *linkname* field, described below, shall not use the *prefix* to produce a pathname. As such, a *linkname* is limited to 100 characters. If the name does not fit in the space provided, *pax* shall

notify the user of the error, and shall not attempt to store the link on the medium.

The *mode* field provides 12 bits encoded in the ISO/IEC 646:1991 standard octal digit representation. The encoded bits shall represent the following values:

Table 4-15 *ustar mode* Field

Bit Value	POSIX.1-2008 Bit	Description
04 000	S_ISUID	Set UID on execution.
02 000	S_ISGID	Set GID on execution.
01 000	<reserved>	Reserved for future standardization.
00 400	S_IRUSR	Read permission for file owner class.
00 200	S_IWUSR	Write permission for file owner class.
00 100	S_IXUSR	Execute/search permission for file owner class.
00 040	S_IRGRP	Read permission for file group class.
00 020	S_IWGRP	Write permission for file group class.
00 010	S_IXGRP	Execute/search permission for file group class.
00 004	S_IROTH	Read permission for file other class.
00 002	S_IWOTH	Write permission for file other class.
00 001	S_IXOTH	Execute/search permission for file other class.

When appropriate privileges are required to set one of these mode bits, and the user restoring the files from the archive does not have appropriate privileges, the mode bits for which the user does not have appropriate privileges shall be ignored. Some of the mode bits in the archive format are not mentioned elsewhere in this volume of POSIX.1-2008. If the implementation does not support those bits, they may be ignored.

The *uid* and *gid* fields are the user and group ID of the owner and group of the file, respectively.

The *size* field is the size of the file in octets. If the *typeflag* field is set to specify a file to be of type 1 (a link) or 2 (a symbolic link), the *size* field shall be specified as zero. If the *typeflag* field is set to specify a file of type 5 (directory), the *size* field shall be interpreted as described under the definition of that record type. No data logical records are stored for types 1, 2, or 5. If the *typeflag* field is set to 3 (character special file), 4 (block special file), or 6 (FIFO), the meaning of the *size* field is unspecified by this volume of POSIX.1-2008, and no data logical records shall be stored on the medium. Additionally, for type 6, the *size* field shall be ignored when reading. If the *typeflag* field is set to any other value, the number of logical records written following the header shall be $(size+511)/512$, ignoring any fraction in the result of the division.

The *mtime* field shall be the modification time of the file at the time it was archived. It is the ISO/IEC 646:1991 standard representation of the octal value of the modification time obtained from the *stat()* function.

The *chksum* field shall be the ISO/IEC 646:1991 standard IRV representation of the octal value of the simple sum of all octets in the header logical record. Each octet in the header shall be treated as an unsigned value. These values shall be added to an unsigned integer, initialized to zero, the precision of which is not less than 17 bits. When calculating the checksum, the *chksum* field is treated as if it were all <space> characters.

The *typeflag* field specifies the type of file archived. If a particular implementation does not recognize the type, or the user does not have appropriate privileges to create that type, the file shall be extracted as if it were a regular file if the file type is defined to have a meaning for the *size* field that could cause data logical records to be written on the medium (see the previous description for *size*). If conversion to a regular file occurs, the *pax* utility shall produce an error indicating that the conversion took place. All of the *typeflag* fields shall be coded in the

100004	ISO/IEC 646: 1991 standard IRV:	
100005	0	Represents a regular file. For backwards-compatibility, a <i>typeflag</i> value of binary zero (' \0 ') should be recognized as meaning a regular file when extracting files from the archive. Archives written with this version of the archive file format create regular files with a <i>typeflag</i> value of the ISO/IEC 646: 1991 standard IRV ' 0 '.
100006		
100007		
100008		
100009	1	Represents a file linked to another file, of any type, previously archived. Such files are identified by having the same device and file serial numbers, and pathnames that refer to different directory entries. All such files shall be archived as linked files. The linked-to name is specified in the <i>linkname</i> field with a NUL-character terminator if it is less than 100 octets in length.
100010		
100011		
100012		
100013		
100014	2	Represents a symbolic link. The contents of the symbolic link shall be stored in the <i>linkname</i> field.
100015		
100016	3, 4	Represent character special files and block special files respectively. In this case the <i>devmajor</i> and <i>devminor</i> fields shall contain information defining the device, the format of which is unspecified by this volume of POSIX.1-2008. Implementations may map the device specifications to their own local specification or may ignore the entry.
100017		
100018		
100019		
100020	5	Specifies a directory or subdirectory. On systems where disk allocation is performed on a directory basis, the <i>size</i> field shall contain the maximum number of octets (which may be rounded to the nearest disk block allocation unit) that the directory may hold. A <i>size</i> field of zero indicates no such limiting. Systems that do not support limiting in this manner should ignore the <i>size</i> field.
100021		
100022		
100023		
100024		
100025	6	Specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence of this file and not its contents.
100026		
100027	7	Reserved to represent a file to which an implementation has associated some high-performance attribute. Implementations without such extensions should treat this file as a regular file (type 0).
100028		
100029		
100030	A–Z	The letters 'A' to 'Z', inclusive, are reserved for custom implementations. All other values are reserved for future versions of this standard.
100031		
100032	It is unspecified whether files with pathnames that refer to the same directory entry are archived as linked files or as separate files. If they are archived as linked files, this means that attempting to extract both pathnames from the resulting archive will always cause an error (unless the <i>–u</i> option is used) because the link cannot be created.	
100033		
100034		
100035		
100036	It is unspecified whether files with the same device and file serial numbers being appended to an archive are treated as linked files to members that were in the archive before the append.	
100037		
100038	Attempts to archive a socket using <i>ustar</i> interchange format shall produce a diagnostic message.	
100039	Handling of other file types is implementation-defined.	
100040	The <i>magic</i> field is the specification that this archive was output in this archive format. If this field contains <i>ustar</i> (the five characters from the ISO/IEC 646: 1991 standard IRV shown followed by NUL), the <i>uname</i> and <i>gname</i> fields shall contain the ISO/IEC 646: 1991 standard IRV representation of the owner and group of the file, respectively (truncated to fit, if necessary).	
100041		
100042		
100043		
100044	When the file is restored by a privileged, protection-preserving version of the utility, the user and group databases shall be scanned for these names. If found, the user and group IDs contained within these files shall be used rather than the values contained within the <i>uid</i> and <i>gid</i> fields.	
100045		
100046		
100047		

cpio Interchange Format

The octet-oriented **cpio** archive format shall be a series of entries, each comprising a header that describes the file, the name of the file, and then the contents of the file.

An archive may be recorded as a series of fixed-size blocks of octets. This blocking shall be used only to make physical I/O more efficient. The last group of blocks shall always be at the full size.

For the octet-oriented **cpio** archive format, the individual entry information shall be in the order indicated and described by the following table; see also the **<cpio.h>** header.

Table 4-16 Octet-Oriented cpio Archive Entry

Header Field Name	Length (in Octets)	Interpreted as
<i>c_magic</i>	6	Octal number
<i>c_dev</i>	6	Octal number
<i>c_ino</i>	6	Octal number
<i>c_mode</i>	6	Octal number
<i>c_uid</i>	6	Octal number
<i>c_gid</i>	6	Octal number
<i>c_nlink</i>	6	Octal number
<i>c_rdev</i>	6	Octal number
<i>c_mtime</i>	11	Octal number
<i>c_namesize</i>	6	Octal number
<i>c_filesize</i>	11	Octal number
Filename Field Name	Length	Interpreted as
<i>c_name</i>	<i>c_namesize</i>	Pathname string
File Data Field Name	Length	Interpreted as
<i>c_filedata</i>	<i>c_filesize</i>	Data

cpio Header

For each file in the archive, a header as defined previously shall be written. The information in the header fields is written as streams of the ISO/IEC 646:1991 standard characters interpreted as octal numbers. The octal numbers shall be extended to the necessary length by appending the ISO/IEC 646:1991 standard IRV zeros at the most-significant-digit end of the number; the result is written to the most-significant digit of the stream of octets first. The fields shall be interpreted as follows:

- c_magic* Identify the archive as being a transportable archive by containing the identifying value "070707".
- c_dev, c_ino* Contains values that uniquely identify the file within the archive (that is, no files contain the same pair of *c_dev* and *c_ino* values unless they are links to the same file). The values shall be determined in an unspecified manner.
- c_mode* Contains the file type and access permissions as defined in the following table.

100086

Table 4-17 Values for `cpio c_mode` Field

100087

100088

100089

100090

100091

100092

100093

100094

100095

100096

100097

100098

100099

100100

100101

100102

100103

100104

100105

100106

100107

100108

File Permissions Name	Value	Indicates
C_IRUSR	000 400	Read by owner
C_IWUSR	000 200	Write by owner
C_IXUSR	000 100	Execute by owner
C_IRGRP	000 040	Read by group
C_IWGRP	000 020	Write by group
C_IXGRP	000 010	Execute by group
C_IROTH	000 004	Read by others
C_IWOTH	000 002	Write by others
C_IXOTH	000 001	Execute by others
C_ISUID	004 000	Set <i>uid</i>
C_ISGID	002 000	Set <i>gid</i>
C_ISVTX	001 000	Reserved
File Type Name	Value	Indicates
C_ISDIR	040 000	Directory
C_ISFIFO	010 000	FIFO
C_ISREG	0100 000	Regular file
C_ISLNK	0120 000	Symbolic link
C_ISBLK	060 000	Block special file
C_ISCHR	020 000	Character special file
C_ISSOCK	0140 000	Socket
C_ISCTG	0110 000	Reserved

100109

100110

100111

100112

100113

Directories, FIFOs, symbolic links, and regular files shall be supported on a system conforming to this volume of POSIX.1-2008; additional values defined previously are reserved for compatibility with existing systems. Additional file types may be supported; however, such files should not be written to archives intended to be transported to other systems.

100114

c_uid

Contains the user ID of the owner.

100115

c_gid

Contains the group ID of the group.

100116

c_nlink

Contains a number greater than or equal to the number of links in the archive referencing the file. If the `-a` option is used to append to a *cpio* archive, then the *pax* utility need not account for the files in the existing part of the archive when calculating the *c_nlink* values for the appended part of the archive, and need not alter the *c_nlink* values in the existing part of the archive if additional files with the same *c_dev* and *c_ino* values are appended to the archive.

100117

100118

100119

100120

100121

100122

c_rdev

Contains implementation-defined information for character or block special files.

100123

100124

c_mtime

Contains the latest time of modification of the file at the time the archive was created.

100125

c_namesize

Contains the length of the pathname, including the terminating NUL character.

100126

c_filesize

Contains the length in octets of the data section following the header structure.

cpio Filename

The *c_name* field shall contain the pathname of the file. The length of this field in octets is the value of *c_namesize*.

If a filename is found on the medium that would create an invalid pathname, it is implementation-defined whether the data from the file is stored on the file hierarchy and under what name it is stored.

All characters shall be represented in the ISO/IEC 646:1991 standard IRV. For maximum portability between implementations, names should be selected from characters represented by the portable filename character set as octets with the most significant bit zero. If an implementation supports the use of characters outside the portable filename character set in names for files, users, and groups, one or more implementation-defined encodings of these characters shall be provided for interchange purposes. However, the *pax* utility shall never create filenames on the local system that cannot be accessed via the procedures described previously in this volume of POSIX.1-2008. If a filename is found on the medium that would create an invalid filename, it is implementation-defined whether the data from the file is stored on the local file system and under what name it is stored. The *pax* utility may choose to ignore these files as long as it produces an error indicating that the file is being ignored.

cpio File Data

Following *c_name*, there shall be *c_filesize* octets of data. Interpretation of such data occurs in a manner dependent on the file. For regular files, the data shall consist of the contents of the file. For symbolic links, the data shall consist of the contents of the symbolic link. If *c_filesize* is zero, no data shall be contained in *c_filedata*.

When restoring from an archive:

- If the user does not have appropriate privileges to create a file of the specified type, *pax* shall ignore the entry and write an error message to standard error.
- Only regular files and symbolic links have data to be restored. Presuming a regular file meets any selection criteria that might be imposed on the format-reading utility by the user, such data shall be restored.
- If a user does not have appropriate privileges to set a particular mode flag, the flag shall be ignored. Some of the mode flags in the archive format are not mentioned elsewhere in this volume of POSIX.1-2008. If the implementation does not support those flags, they may be ignored.

cpio Special Entries

FIFO special files, directories, and the trailer shall be recorded with *c_filesize* equal to zero. Symbolic links shall be recorded with *c_filesize* equal to the length of the contents of the symbolic link. For other special files, *c_filesize* is unspecified by this volume of POSIX.1-2008. The header for the next file entry in the archive shall be written directly after the last octet of the file entry preceding it. A header denoting the filename **TRAILER!!!** shall indicate the end of the archive; the contents of octets in the last block of the archive following such a header are undefined.

EXIT STATUS

The following exit values shall be returned:

- 0 All files were processed successfully.

100169 >0 An error occurred.

100170 CONSEQUENCES OF ERRORS

100171 If *pax* cannot create a file or a link when reading an archive or cannot find a file when writing an
 100172 archive, or cannot preserve the user ID, group ID, or file mode when the *-p* option is specified, a
 100173 diagnostic message shall be written to standard error and a non-zero exit status shall be
 100174 returned, but processing shall continue. In the case where *pax* cannot create a link to a file, *pax*
 100175 shall not, by default, create a second copy of the file.

100176 If the extraction of a file from an archive is prematurely terminated by a signal or error, *pax* may
 100177 have only partially extracted the file or (if the *-n* option was not specified) may have extracted a
 100178 file of the same name as that specified by the user, but which is not the file the user wanted.
 100179 Additionally, the file modes of extracted directories may have additional bits from the S_IRWXU
 100180 mask set as well as incorrect modification and access times.

100181 APPLICATION USAGE

100182 Caution is advised when using the *-a* option to append to a *cpio* format archive. If any of the
 100183 files being appended happen to be given the same *c_dev* and *c_ino* values as a file in the existing
 100184 part of the archive, then they may be treated as links to that file on extraction. Thus, it is risky to
 100185 use *-a* with *cpio* format except when it is done on the same system that the original archive was
 100186 created on, and with the same *pax* utility, and in the knowledge that there has been little or no
 100187 file system activity since the original archive was created that could lead to any of the files
 100188 appended being given the same *c_dev* and *c_ino* values as an unrelated file in the existing part of
 100189 the archive. Also, when (intentionally) appending additional links to a file in the existing part of
 100190 the archive, the *c_nlink* values in the modified archive can be smaller than the number of links to
 100191 the file in the archive, which may mean that the links are not preserved on extraction.

100192 The *-p* (privileges) option was invented to reconcile differences between historical *tar* and *cpio*
 100193 implementations. In particular, the two utilities use *-m* in diametrically opposed ways. The *-p*
 100194 option also provides a consistent means of extending the ways in which future file attributes can
 100195 be addressed, such as for enhanced security systems or high-performance files. Although it may
 100196 seem complex, there are really two modes that are most commonly used:

100197 *-p e* "Preserve everything". This would be used by the historical superuser, someone with
 100198 all appropriate privileges, to preserve all aspects of the files as they are recorded in the
 100199 archive. The *e* flag is the sum of *o* and *p*, and other implementation-defined attributes.

100200 *-p p* "Preserve" the file mode bits. This would be used by the user with regular privileges
 100201 who wished to preserve aspects of the file other than the ownership. The file times are
 100202 preserved by default, but two other flags are offered to disable these and use the time
 100203 of extraction.

100204 The one pathname per line format of standard input precludes pathnames containing <newline>
 100205 characters. Although such pathnames violate the portable filename guidelines, they may exist
 100206 and their presence may inhibit usage of *pax* within shell scripts. This problem is inherited from
 100207 historical archive programs. The problem can be avoided by listing filename arguments on the
 100208 command line instead of on standard input.

100209 It is almost certain that appropriate privileges are required for *pax* to accomplish parts of this
 100210 volume of POSIX.1-2008. Specifically, creating files of type block special or character special,
 100211 restoring file access times unless the files are owned by the user (the *-t* option), or preserving file
 100212 owner, group, and mode (the *-p* option) all probably require appropriate privileges.

100213 In *read* mode, implementations are permitted to overwrite files when the archive has multiple
 100214 members with the same name. This may fail if permissions on the first version of the file do not
 100215 permit it to be overwritten.

The **cpio** and **ustar** formats can only support files up to 8 589 934 592 bytes ($8 * 2^{30}$) in size.

When archives containing binary header information are listed, the filenames printed may cause strange behavior on some terminals.

EXAMPLES

The following command:

```
pax -w -f /dev/rmt/lm .
```

copies the contents of the current directory to tape drive 1, medium density (assuming historical System V device naming procedures—the historical BSD device name would be **/dev/rmt9**).

The following commands:

```
mkdir newdir
pax -rw olddir newdir
```

copy the **olddir** directory hierarchy to **newdir**.

```
pax -r -s ' , ^//*usr//* , , ' -f a.pax
```

reads the archive **a.pax**, with all files rooted in **/usr** in the archive extracted relative to the current directory.

Using the option:

```
-o listopt="%M %(atime)T %(size)D %(name)s"
```

overrides the default output description in Standard Output and instead writes:

```
-rw-rw--- Jan 12 15:53 2003 1492 /usr/foo/bar
```

Using the options:

```
-o listopt='%L\t%(size)D\n%>' \
-o listopt='(name)s\n%(atime)T\n%T'
```

overrides the default output description in Standard Output and instead writes:

```
/usr/foo/bar -> /tmp 1492
/usr/fo
Jan 12 15:53 1991
Jan 31 15:53 2003
```

RATIONALE

The **pax** utility was new for the ISO POSIX-2: 1993 standard. It represents a peaceful compromise between advocates of the historical **tar** and **cpio** utilities.

A fundamental difference between **cpio** and **tar** was in the way directories were treated. The **cpio** utility did not treat directories differently from other files, and to select a directory and its contents required that each file in the hierarchy be explicitly specified. For **tar**, a directory matched every file in the file hierarchy it rooted.

The **pax** utility offers both interfaces; by default, directories map into the file hierarchy they root. The **-d** option causes **pax** to skip any file not explicitly referenced, as **cpio** historically did. The **tar** **-style** behavior was chosen as the default because it was believed that this was the more common usage and because **tar** is the more commonly available interface, as it was historically provided on both System V and BSD implementations.

The data interchange format specification in this volume of POSIX.1-2008 requires that processes with “appropriate privileges” shall always restore the ownership and permissions of extracted

files exactly as archived. If viewed from the historic equivalence between superuser and “appropriate privileges”, there are two problems with this requirement. First, users running as superusers may unknowingly set dangerous permissions on extracted files. Second, it is needlessly limiting, in that superusers cannot extract files and own them as superuser unless the archive was created by the superuser. (It should be noted that restoration of ownerships and permissions for the superuser, by default, is historical practice in *cpio*, but not in *tar*.) In order to avoid these two problems, the *pax* specification has an additional “privilege” mechanism, the **-p** option. Only a *pax* invocation with the privileges needed, and which has the **-p** option set using the **e** specification character, has appropriate privileges to restore full ownership and permission information.

Note also that this volume of POSIX.1-2008 requires that the file ownership and access permissions shall be set, on extraction, in the same fashion as the *creat()* function when provided with the mode stored in the archive. This means that the file creation mask of the user is applied to the file permissions.

Users should note that directories may be created by *pax* while extracting files with permissions that are different from those that existed at the time the archive was created. When extracting sensitive information into a directory hierarchy that no longer exists, users are encouraged to set their file creation mask appropriately to protect these files during extraction.

The table of contents output is written to standard output to facilitate pipeline processing.

An early proposal had hard links displaying for all pathnames. This was removed because it complicates the output of the case where **-v** is not specified and does not match historical *cpio* usage. The hard-link information is available in the **-v** display.

The description of the **-l** option allows implementations to make hard links to symbolic links. POSIX.1-2008 does not specify any way to create a hard link to a symbolic link, but many implementations provide this capability as an extension. If there are hard links to symbolic links when an archive is created, the implementation is required to archive the hard link in the archive (unless **-H** or **-L** is specified). When in **read** mode and in **copy** mode, implementations supporting hard links to symbolic links should use them when appropriate.

The archive formats inherited from the POSIX.1-1990 standard have certain restrictions that have been brought along from historical usage. For example, there are restrictions on the length of pathnames stored in the archive. When *pax* is used in **copy(-rw)** mode (copying directory hierarchies), the ability to use extensions from the **-xpax** format overcomes these restrictions.

The default *blocksize* value of 5 120 bytes for *cpio* was selected because it is one of the standard block-size values for *cpio*, set when the **-B** option is specified. (The other default block-size value for *cpio* is 512 bytes, and this was considered to be too small.) The default block value of 10 240 bytes for *tar* was selected because that is the standard block-size value for BSD *tar*. The maximum block size of 32 256 bytes (2^{15} –512 bytes) is the largest multiple of 512 bytes that fits into a signed 16-bit tape controller transfer register. There are known limitations in some historical systems that would prevent larger blocks from being accepted. Historical values were chosen to improve compatibility with historical scripts using *dd* or similar utilities to manipulate archives. Also, default block sizes for any file type other than character special file has been deleted from this volume of POSIX.1-2008 as unimportant and not likely to affect the structure of the resulting archive.

Implementations are permitted to modify the block-size value based on the archive format or the device to which the archive is being written. This is to provide implementations with the opportunity to take advantage of special types of devices, and it should not be used without a great deal of consideration as it almost certainly decreases archive portability.

The intended use of the **-n** option was to permit extraction of one or more files from the archive without processing the entire archive. This was viewed by the standard developers as offering significant performance advantages over historical implementations. The **-n** option in early proposals had three effects; the first was to cause special characters in patterns to not be treated specially. The second was to cause only the first file that matched a pattern to be extracted. The third was to cause *pax* to write a diagnostic message to standard error when no file was found matching a specified pattern. Only the second behavior is retained by this volume of POSIX.1-2008, for many reasons. First, it is in general not acceptable for a single option to have multiple effects. Second, the ability to make pattern matching characters act as normal characters is useful for parts of *pax* other than file extraction. Third, a finer degree of control over the special characters is useful because users may wish to normalize only a single special character in a single filename. Fourth, given a more general escape mechanism, the previous behavior of the **-n** option can be easily obtained using the **-s** option or a *sed* script. Finally, writing a diagnostic message when a pattern specified by the user is unmatched by any file is useful behavior in all cases.

In this version, the **-n** was removed from the **copy** mode synopsis of *pax*; it is inapplicable because there are no pattern operands specified in this mode.

There is another method than *pax* for copying subtrees in POSIX.1-2008 described as part of the *cp* utility. Both methods are historical practice: *cp* provides a simpler, more intuitive interface, while *pax* offers a finer granularity of control. Each provides additional functionality to the other; in particular, *pax* maintains the hard-link structure of the hierarchy while *cp* does not. It is the intention of the standard developers that the results be similar (using appropriate option combinations in both utilities). The results are not required to be identical; there seemed insufficient gain to applications to balance the difficulty of implementations having to guarantee that the results would be exactly identical.

A single archive may span more than one file. It is suggested that implementations provide informative messages to the user on standard error whenever the archive file is changed.

The **-d** option (do not create intermediate directories not listed in the archive) found in early proposals was originally provided as a complement to the historic **-d** option of *cpio*. It has been deleted.

The **-s** option in early proposals specified a subset of the substitution command from the *ed* utility. As there was no reason for only a subset to be supported, the **-s** option is now compatible with the current *ed* specification. Since the delimiter can be any non-null character, the following usage with single `<space>` characters is valid:

```
pax -s " foo bar " ...
```

The **-t** description is worded so as to note that this may cause the access time update caused by some other activity (which occurs while the file is being read) to be overwritten.

The default behavior of *pax* with regard to file modification times is the same as historical implementations of *tar*. It is not the historical behavior of *cpio*.

Because the **-i** option uses **/dev/tty**, utilities without a controlling terminal are not able to use this option.

The **-y** option, found in early proposals, has been deleted because a line containing a single `<period>` for the **-i** option has equivalent functionality. The special lines for the **-i** option (a single `<period>` and the empty line) are historical practice in *cpio*.

In early drafts, a **-echarmap** option was included to increase portability of files between systems using different coded character sets. This option was omitted because it was apparent that

consensus could not be formed for it. In this version, the use of UTF-8 should be an adequate substitute.

The ISO POSIX-2:1993 standard and ISO POSIX-1 standard requirements for *pax*, however, made it very difficult to create a single archive containing files created using extended characters provided by different locales. This version adds the **hdrcharset** keyword to make it possible to archive files in these cases without dropping files due to translation errors.

Translating filenames and other attributes from a locale's encoding to UTF-8 and then back again can lose information, as the resulting filename might not be byte-for-byte equivalent to the original. To avoid this problem, users can specify the **-o hdrcharset=binary** option, which will cause the resulting archive to use binary format for all names and attributes. Such archives are not portable among hosts that use different native encodings (e.g., EBCDIC *versus* ASCII-based encodings), but they will allow interchange among the vast majority of POSIX file systems in practical use. Also, the **-o hdrcharset=binary** option will cause *pax* in **copy** mode to behave more like other standard utilities such as *cp*.

If the values specified by the **-o exthdr.name=value**, **-o globexthdr.name=value**, or by **\$TMPDIR** (if **-o globexthdr.name** is not specified) require a character encoding other than that described in the ISO/IEC 646:1991 standard, a **path** extended header record will have to be created for the file. If a **hdrcharset** extended header record is active for such headers, it will determine the codeset used for the value field in these extended **path** header records. These **path** extended header records always need to be created when writing an archive even if **hdrcharset=binary** has been specified and would contain the same (binary) data that appears in the **ustar** header record prefix and *name* fields. (In other words, an extended header **path** record is always required to be generated if the *prefix* or *name* fields contain non-ASCII characters even when **hdrcharset=binary** is also in effect for that file.)

The **-k** option was added to address international concerns about the dangers involved in the character set transformations of **-e** (if the target character set were different from the source, the filenames might be transformed into names matching existing files) and also was made more general to protect files transferred between file systems with different {NAME_MAX} values (truncating a filename on a smaller system might also inadvertently overwrite existing files). As stated, it prevents any overwriting, even if the target file is older than the source. This version adds more granularity of options to solve this problem by introducing the **-o invalid=option**—specifically the **UTF-8** and **binary** actions. (Note that an existing file is still subject to overwriting in this case. The **-k** option closes that loophole.)

Some of the file characteristics referenced in this volume of POSIX.1-2008 might not be supported by some archive formats. For example, neither the **tar** nor **cpio** formats contain the file access time. For this reason, the **e** specification character has been provided, intended to cause all file characteristics specified in the archive to be retained.

It is required that extracted directories, by default, have their access and modification times and permissions set to the values specified in the archive. This has obvious problems in that the directories are almost certainly modified after being extracted and that directory permissions may not permit file creation. One possible solution is to create directories with the mode specified in the archive, as modified by the *umask* of the user, with sufficient permissions to allow file creation. After all files have been extracted, *pax* would then reset the access and modification times and permissions as necessary.

The list-mode formatting description borrows heavily from the one defined by the *printf* utility. However, since there is no separate operand list to get conversion arguments, the format was extended to allow specifying the name of the conversion argument as part of the conversion specification.

The **T** conversion specifier allows time fields to be displayed in any of the date formats. Unlike the *ls* utility, *pax* does not adjust the format when the date is less than six months in the past. This makes parsing the output more predictable.

The **D** conversion specifier handles the ability to display the major/minor or file size, as with *ls*, by using `%-8 (size)D`.

The **L** conversion specifier handles the *ls* display for symbolic links.

Conversion specifiers were added to generate existing known types used for *ls*.

pax Interchange Format

The new POSIX data interchange format was developed primarily to satisfy international concerns that the **ustar** and **cpio** formats did not provide for file, user, and group names encoded in characters outside a subset of the ISO/IEC 646:1991 standard. The standard developers realized that this new POSIX data interchange format should be very extensible because there were other requirements they foresaw in the near future:

- Support international character encodings and locale information
- Support security information (ACLs, and so on)
- Support future file types, such as realtime or contiguous files
- Include data areas for implementation use
- Support systems with words larger than 32 bits and timers with subsecond granularity

The following were not goals for this format because these are better handled by separate utilities or are inappropriate for a portable format:

- Encryption
- Compression
- Data translation between locales and codesets
- *inode* storage

The format chosen to support the goals is an extension of the **ustar** format. Of the two formats previously available, only the **ustar** format was selected for extensions because:

- It was easier to extend in an upwards-compatible way. It offered version flags and header block type fields with room for future standardization. The **cpio** format, while possessing a more flexible file naming methodology, could not be extended without breaking some theoretical implementation or using a dummy filename that could be a legitimate filename.
- Industry experience since the original “tar wars” fought in developing the ISO POSIX-1 standard has clearly been in favor of the **ustar** format, which is generally the default output format selected for *pax* implementations on new systems.

The new format was designed with one additional goal in mind: reasonable behavior when an older *tar* or *pax* utility happened to read an archive. Since the POSIX.1-1990 standard mandated that a “format-reading utility” had to treat unrecognized *typeflag* values as regular files, this allowed the format to include all the extended information in a pseudo-regular file that preceded each real file. An option is given that allows the archive creator to set up reasonable names for these files on the older systems. Also, the normative text suggests that reasonable file access values be used for this **ustar** header block. Making these header files inaccessible for convenient reading and deleting would not be reasonable. File permissions of 600 or 700 are suggested.

The **ustar** *typeflag* field was used to accommodate the additional functionality of the new format rather than magic or version because the POSIX.1-1990 standard (and, by reference, the previous version of *pax*), mandated the behavior of the format-reading utility when it encountered an unknown *typeflag*, but was silent about the other two fields.

Early proposals for the first version of this standard contained a proposed archive format that was based on compatibility with the standard for tape files (ISO 1001, similar to the format used historically on many mainframes and minicomputers). This format was overly complex and required considerable overhead in volume and header records. Furthermore, the standard developers felt that it would not be acceptable to the community of POSIX developers, so it was later changed to be a format more closely related to historical practice on POSIX systems.

The prefix and name split of pathnames in **ustar** was replaced by the single path extended header record for simplicity.

The concept of a global extended header (*typeflag*g) was controversial. If this were applied to an archive being recorded on magnetic tape, a few unreadable blocks at the beginning of the tape could be a serious problem; a utility attempting to extract as many files as possible from a damaged archive could lose a large percentage of file header information in this case. However, if the archive were on a reliable medium, such as a CD-ROM, the global extended header offers considerable potential size reductions by eliminating redundant information. Thus, the text warns against using the global method for unreliable media and provides a method for implanting global information in the extended header for each file, rather than in the *typeflag* g records.

No facility for data translation or filtering on a per-file basis is included because the standard developers could not invent an interface that would allow this in an efficient manner. If a filter, such as encryption or compression, is to be applied to all the files, it is more efficient to apply the filter to the entire archive as a single file. The standard developers considered interfaces that would invoke a shell script for each file going into or out of the archive, but the system overhead in this approach was considered to be too high.

One such approach would be to have **filter=** records that give a pathname for an executable. When the program is invoked, the file and archive would be open for standard input/output and all the header fields would be available as environment variables or command-line arguments. The standard developers did discuss such schemes, but they were omitted from POSIX.1-2008 due to concerns about excessive overhead. Also, the program itself would need to be in the archive if it were to be used portably.

There is currently no portable means of identifying the character set(s) used for a file in the file system. Therefore, *pax* has not been given a mechanism to generate charset records automatically. The only portable means of doing this is for the user to write the archive using the **-ocharset=string** command line option. This assumes that all of the files in the archive use the same encoding. The “implementation-defined” text is included to allow for a system that can identify the encodings used for each of its files.

The table of standards that accompanies the charset record description is acknowledged to be very limited. Only a limited number of character set standards is reasonable for maximal interchange. Any character set is, of course, possible by prior agreement. It was suggested that EBCDIC be listed, but it was omitted because it is not defined by a formal standard. Formal standards, and then only those with reasonably large followings, can be included here, simply as a matter of practicality. The <value>s represent names of officially registered character sets in the format required by the ISO 2375:1985 standard.

The normal <comma> or <blank>-separated list rules are not followed in the case of keyword options to allow ease of argument parsing for *getopts*.

Further information on character encodings is in [pax Archive Character Set Encoding/Decoding](#) (on page 3038).

The standard developers have reserved keyword name space for vendor extensions. It is suggested that the format to be used is:

VENDOR.keyword

where *VENDOR* is the name of the vendor or organization in all uppercase letters. It is further suggested that the keyword following the <period> be named differently than any of the standard keywords so that it could be used for future standardization, if appropriate, by omitting the *VENDOR* prefix.

The <length> field in the extended header record was included to make it simpler to step through the records, even if a record contains an unknown format (to a particular *pax*) with complex interactions of special characters. It also provides a minor integrity checkpoint within the records to aid a program attempting to recover files from a damaged archive.

There are no extended header versions of the *devmajor* and *devminor* fields because the unspecified format **ustar** header field should be sufficient. If they are not, vendor-specific extended keywords (such as *VENDOR.devmajor*) should be used.

Device and *i*-number labeling of files was not adopted from *cpio*; files are interchanged strictly on a symbolic name basis, as in **ustar**.

Just as with the **ustar** format descriptions, the new format makes no special arrangements for multi-volume archives. Each of the *pax* archive types is assumed to be inside a single POSIX file and splitting that file over multiple volumes (diskettes, tape cartridges, and so on), processing their labels, and mounting each in the proper sequence are considered to be implementation details that cannot be described portably.

The **pax** format is intended for interchange, not only for backup on a single (family of) systems. It is not as densely packed as might be possible for backup:

- It contains information as coded characters that could be coded in binary.
- It identifies extended records with name fields that could be omitted in favor of a fixed-field layout.
- It translates names into a portable character set and identifies locale-related information, both of which are probably unnecessary for backup.

The requirements on restoring from an archive are slightly different from the historical wording, allowing for non-monolithic privilege to bring forward as much as possible. In particular, attributes such as “high performance file” might be broadly but not universally granted while set-user-ID or *chown()* might be much more restricted. There is no implication in POSIX.1-2008 that the security information be honored after it is restored to the file hierarchy, in spite of what might be improperly inferred by the silence on that topic. That is a topic for another standard.

Links are recorded in the fashion described here because a link can be to any file type. It is desirable in general to be able to restore part of an archive selectively and restore all of those files completely. If the data is not associated with each link, it is not possible to do this. However, the data associated with a file can be large, and when selective restoration is not needed, this can be a significant burden. The archive is structured so that files that have no associated data can always be restored by the name of any link name of any link, and the user may choose whether data is recorded with each instance of a file that contains data. The format permits mixing of both types of links in a single archive; this can be done for special needs, and *pax* is expected to interpret such archives on input properly, despite the fact that there is no *pax* option that would

force this mixed case on output. (When **-o linkdata** is used, the output must contain the duplicate data, but the implementation is free to include it or omit it when **-o linkdata** is not used.)

The time values are included as extended header records for those implementations needing more than the eleven octal digits allowed by the **ustar** format. Portable file timestamps cannot be negative. If *pax* encounters a file with a negative timestamp in **copy** or **write** mode, it can reject the file, substitute a non-negative timestamp, or generate a non-portable timestamp with a leading '-'. Even though some implementations can support finer file-time granularities than seconds, the normative text requires support only for seconds since the Epoch because the ISO POSIX-1 standard states them that way. The **ustar** format includes only *mtime*; the new format adds *atime* and *ctime* for symmetry. The *atime* access time restored to the file system will be affected by the **-p a** and **-p e** options. The *ctime* creation time (actually *inode* modification time) is described with appropriate privileges so that it can be ignored when writing to the file system. POSIX does not provide a portable means to change file creation time. Nothing is intended to prevent a non-portable implementation of *pax* from restoring the value.

The *gid*, *size*, and *uid* extended header records were included to allow expansion beyond the sizes specified in the regular *tar* header. New file system architectures are emerging that will exhaust the 12-digit size field. There are probably not many systems requiring more than 8 digits for user and group IDs, but the extended header values were included for completeness, allowing overrides for all of the decimal values in the *tar* header.

The standard developers intended to describe the effective results of *pax* with regard to file ownerships and permissions; implementations are not restricted in timing or sequencing the restoration of such, provided the results are as specified.

Much of the text describing the extended headers refers to use in “**write** or **copy** modes”. The **copy** mode references are due to the normative text: “The effect of the copy shall be as if the copied files were written to an archive file and then subsequently extracted ...”. There is certainly no way to test whether *pax* is actually generating the extended headers in **copy** mode, but the effects must be as if it had.

pax Archive Character Set Encoding/Decoding

There is a need to exchange archives of files between systems of different native codesets. Filenames, group names, and user names must be preserved to the fullest extent possible when an archive is read on the receiving platform. Translation of the contents of files is not within the scope of the *pax* utility.

There will also be the need to represent characters that are not available on the receiving platform. These unsupported characters cannot be automatically folded to the local set of characters due to the chance of collisions. This could result in overwriting previous extracted files from the archive or pre-existing files on the system.

For these reasons, the codeset used to represent characters within the extended header records of the *pax* archive must be sufficiently rich to handle all commonly used character sets. The fields requiring translation include, at a minimum, filenames, user names, group names, and link pathnames. Implementations may wish to have localized extended keywords that use non-portable characters.

The standard developers considered the following options:

- The archive creator specifies the well-defined name of the source codeset. The receiver must then recognize the codeset name and perform the appropriate translations to the destination codeset.

- The archive creator includes within the archive the character mapping table for the source codeset used to encode extended header records. The receiver must then read the character mapping table and perform the appropriate translations to the destination codeset.
- The archive creator translates the extended header records in the source codeset into a canonical form. The receiver must then perform the appropriate translations to the destination codeset.

The approach that incorporates the name of the source codeset poses the problem of codeset name registration, and makes the archive useless to *pax* archive decoders that do not recognize that codeset.

Because parts of an archive may be corrupted, the standard developers felt that including the character map of the source codeset was too fragile. The loss of this one key component could result in making the entire archive useless. (The difference between this and the global extended header decision was that the latter has a workaround—duplicating extended header records on unreliable media—but this would be too burdensome for large character set maps.)

Both of the above approaches also put an undue burden on the *pax* archive receiver to handle the cross-product of all source and destination codesets.

To simplify the translation from the source codeset to the canonical form and from the canonical form to the destination codeset, the standard developers decided that the internal representation should be a stateless encoding. A stateless encoding is one where each codepoint has the same meaning, without regard to the decoder being in a specific state. An example of a stateful encoding would be the Japanese Shift-JIS; an example of a stateless encoding would be the ISO/IEC 646:1991 standard (equivalent to 7-bit ASCII).

For these reasons, the standard developers decided to adopt a canonical format for the representation of file information strings. The obvious, well-endorsed candidate is the ISO/IEC 10646-1:2000 standard (based in part on Unicode), which can be used to represent the characters of virtually all standardized character sets. The standard developers initially agreed upon using UCS2 (16-bit Unicode) as the internal representation. This repertoire of characters provides a sufficiently rich set to represent all commonly-used codesets.

However, the standard developers found that the 16-bit Unicode representation had some problems. It forced the issue of standardizing byte ordering. The 2-byte length of each character made the extended header records twice as long for the case of strings coded entirely from historical 7-bit ASCII. For these reasons, the standard developers chose the UTF-8 defined in the ISO/IEC 10646-1:2000 standard. This multi-byte representation encodes UCS2 or UCS4 characters reliably and deterministically, eliminating the need for a canonical byte ordering. In addition, NUL octets and other characters possibly confusing to POSIX file systems do not appear, except to represent themselves. It was realized that certain national codesets take up more space after the encoding, due to their placement within the UCS range; it was felt that the usefulness of the encoding of the names outweighs the disadvantage of size increase for file, user, and group names.

The encoding of UTF-8 is as follows:

UCS4 Hex Encoding	UTF-8 Binary Encoding
00000000–0000007F	0xxxxxxx
00000080–000007FF	110xxxxx 10xxxxxx
00000800–0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
00010000–001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
00200000–03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

100626 04000000-7FFFFFFF 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
 100627 where each 'x' represents a bit value from the character being translated.

100628 **ustar Interchange Format**

100629 The description of the **ustar** format reflects numerous enhancements over pre-1988 versions of
 100630 the historical *tar* utility. The goal of these changes was not only to provide the functional
 100631 enhancements desired, but also to retain compatibility between new and old versions. This
 100632 compatibility has been retained. Archives written using the old archive format are compatible
 100633 with the new format.

100634 Implementors should be aware that the previous file format did not include a mechanism to
 100635 archive directory type files. For this reason, the convention of using a filename ending with
 100636 <slash> was adopted to specify a directory on the archive.

100637 The total size of the *name* and *prefix* fields have been set to meet the minimum requirements for
 100638 {PATH_MAX}. If a pathname will fit within the *name* field, it is recommended that the pathname
 100639 be stored there without the use of the *prefix* field. Although the *name* field is known to be too
 100640 small to contain {PATH_MAX} characters, the value was not changed in this version of the
 100641 archive file format to retain backwards-compatibility, and instead the *prefix* was introduced.
 100642 Also, because of the earlier version of the format, there is no way to remove the restriction on the
 100643 *linkname* field being limited in size to just that of the *name* field.

100644 The *size* field is required to be meaningful in all implementation extensions, although it could be
 100645 zero. This is required so that the data blocks can always be properly counted.

100646 It is suggested that if device special files need to be represented that cannot be represented in the
 100647 standard format, that one of the extension types (**A-Z**) be used, and that the additional
 100648 information for the special file be represented as data and be reflected in the *size* field.

100649 Attempting to restore a special file type, where it is converted to ordinary data and conflicts with
 100650 an existing filename, need not be specially detected by the utility. If run as an ordinary user, *pax*
 100651 should not be able to overwrite the entries in, for example, */dev* in any case (whether the file is
 100652 converted to another type or not). If run as a privileged user, it should be able to do so, and it
 100653 would be considered a bug if it did not. The same is true of ordinary data files and similarly
 100654 named special files; it is impossible to anticipate the needs of the user (who could really intend
 100655 to overwrite the file), so the behavior should be predictable (and thus regular) and rely on the
 100656 protection system as required.

100657 The value 7 in the *typeflag* field is intended to define how contiguous files can be stored in a
 100658 **ustar** archive. POSIX.1-2008 does not require the contiguous file extension, but does define a
 100659 standard way of archiving such files so that all conforming systems can interpret these file types
 100660 in a meaningful and consistent manner. On a system that does not support extended file types,
 100661 the *pax* utility should do the best it can with the file and go on to the next.

100662 The file protection modes are those conventionally used by the *ls* utility. This is extended beyond
 100663 the usage in the ISO POSIX-2 standard to support the "shared text" or "sticky" bit. It is intended
 100664 that the conformance document should not document anything beyond the existence of and
 100665 support of such a mode. Further extensions are expected to these bits, particularly with
 100666 overloading the set-user-ID and set-group-ID flags.

cpio Interchange Format

The reference to appropriate privileges in the **cpio** format refers to an error on standard output; the **ustar** format does not make comparable statements.

The model for this format was the historical System V *cpio*-c data interchange format. This model documents the portable version of the **cpio** format and not the binary version. It has the flexibility to transfer data of any type described within POSIX.1-2008, yet is extensible to transfer data types specific to extensions beyond POSIX.1-2008 (for example, contiguous files). Because it describes existing practice, there is no question of maintaining upwards-compatibility.

cpio Header

There has been some concern that the size of the *c_ino* field of the header is too small to handle those systems that have very large *inode* numbers. However, the *c_ino* field in the header is used strictly as a hard-link resolution mechanism for archives. It is not necessarily the same value as the *inode* number of the file in the location from which that file is extracted.

The name *c_magic* is based on historical usage.

cpio Filename

For most historical implementations of the *cpio* utility, {PATH_MAX} octets can be used to describe the pathname without the addition of any other header fields (the NUL character would be included in this count). {PATH_MAX} is the minimum value for pathname size, documented as 256 bytes. However, an implementation may use *c_namesize* to determine the exact length of the pathname. With the current description of the **<cpio.h>** header, this pathname size can be as large as a number that is described in six octal digits.

Two values are documented under the *c_mode* field values to provide for extensibility for known file types:

0110 000 Reserved for contiguous files. The implementation may treat the rest of the information for this archive like a regular file. If this file type is undefined, the implementation may create the file as a regular file.

This provides for extensibility of the **cpio** format while allowing for the ability to read old archives. Files of an unknown type may be read as “regular files” on some implementations. On a system that does not support extended file types, the *pax* utility should do the best it can with the file and go on to the next.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2 (on page 2297), *cp*, *ed*, *getopts*, *ls*, *printf*

XBD Section 3.169 (on page 60), Chapter 5 (on page 121), Chapter 8 (on page 173), Section 12.2 (on page 215), **<cpio.h>**

XSH *chown()*, *creat()*, *fstatat()*, *mkdir()*, *mkfifo()*, *utime()*, *write()*

CHANGE HISTORY

First released in Issue 4.

Issue 5

A note is added to the APPLICATION USAGE indicating that the **cpio** and **tar** formats can only support files up to 8 gigabytes in size.

Issue 6

The *pax* utility is aligned with the IEEE P1003.2b draft standard:

- Support has been added for symbolic links in the options and interchange formats.
- A new format has been devised, based on extensions to **ustar**.
- References to the “extended” **tar** and **cpio** formats derived from the POSIX.1-1990 standard have been changed to remove the “extended” adjective because this could cause confusion with the extended *tar* header added in this version. (All references to *tar* are actually to **ustar**.)

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

IEEE PASC Interpretation 1003.2 #168 is applied, clarifying that *mkdir()* and *mkfifo()* calls can ignore an [EEXIST] error when extracting an archive.

IEEE PASC Interpretation 1003.2 #180 is applied, clarifying how extracted files are created when in **read** mode.

IEEE PASC Interpretation 1003.2 #181 is applied, clarifying the description of the **-t** option.

IEEE PASC Interpretation 1003.2 #195 is applied.

IEEE PASC Interpretation 1003.2 #206 is applied, clarifying the handling of links for the **-H**, **-L**, and **-I** options.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/35 is applied, adding the process ID of the *pax* process into certain fields. This change provides a method for the implementation to ensure that different instances of *pax* extracting a file named */a/b/foo* will not collide when processing the extended header information associated with **foo**.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/36 is applied, changing **-x B** to **-x pax** in the OPTIONS section.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/20 is applied, updating the SYNOPSIS to be consistent with the normative text.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/21 is applied, updating the DESCRIPTION to describe the behavior when files to be linked are symbolic links and the system is not capable of making hard links to symbolic links.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/22 is applied, updating the OPTIONS section to describe the behavior for how multiple **-odelete=pattern** options are to be handled.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/23 is applied, updating the **write** option within the OPTIONS section.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/24 is applied, adding a paragraph into the OPTIONS section that states that specifying more than one of the mutually-exclusive options (**-H** and **-L**) is not considered an error and that the last option specified will determine the behavior of the utility.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/25 is applied, removing the *ctime* paragraph within the EXTENDED DESCRIPTION. There is a contradiction in the definition of the *ctime* keyword for the *pax* extended header, in that the *st_ctime* member of the **stat** structure does not refer to a file creation time. No field in the standard **stat** structure from **<sys/stat.h>**

- 100749 includes a file creation time.
- 100750 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/26 is applied, making it clear that *typeflag*
- 100751 1 (**ustar** Interchange Format) applies not only to files that are hard-linked, but also to files that
- 100752 are aliased via symbolic links.
- 100753 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/27 is applied, clarifying the *cpio c_nlink*
- 100754 field.
- 100755 **Issue 7**
- 100756 Austin Group Interpretations 1003.1-2001 #011, #036, #086, and #109 are applied.
- 100757 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the
- 100758 *LC_MESSAGES* environment variable.
- 100759 SD5-XCU-ERN-2 is applied, making **-c** and **-n** mutually-exclusive in the SYNOPSIS.
- 100760 SD5-XCU-ERN-3 is applied, revising the default behavior of **-H** and **-L**.
- 100761 SD5-XCU-ERN-5, SD5-XCU-ERN-6, SD5-XCU-ERN-7, SD5-XCU-ERN-60 are applied.
- 100762 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 100763 The *pax* utility is no longer allowed to create separate identical symbolic links when extracting
- 100764 linked symbolic links from an archive.

pr*Utilities*100765 **NAME**100766 **pr** — print files100767 **SYNOPSIS**

100768 **pr** [+*page*] [-*column*] [-adFmrt] [-e[*char*][*gap*]] [-h *header*] [-i[*char*][*gap*]]
 100769 XSI [-l *lines*] [-n[*char*][*width*]] [-o *offset*] [-s[*char*]] [-w *width*] [-ip]
 100770 [*file...*]

100771 **DESCRIPTION**

100772 The *pr* utility is a printing and pagination filter. If multiple input files are specified, each shall be
 100773 read, formatted, and written to standard output. By default, the input shall be separated into
 100774 66-line pages, each with:

- 100775 • A 5-line header that includes the page number, date, time, and the pathname of the file
- 100776 • A 5-line trailer consisting of blank lines

100777 If standard output is associated with a terminal, diagnostic messages shall be deferred until the
 100778 *pr* utility has completed processing.

100779 When options specifying multi-column output are specified, output text columns shall be of
 100780 equal width; input lines that do not fit into a text column shall be truncated. By default, text
 100781 columns shall be separated with at least one <blank>.

100782 **OPTIONS**

100783 The *pr* utility shall conform to XBD Section 12.2 (on page 215), except that: the *page* option has a
 100784 '+' delimiter; *page* and *column* can be multi-digit numbers; some of the option-arguments are
 100785 optional; and some of the option-arguments cannot be specified as separate arguments from the
 100786 preceding option letter. In particular, the *-s* option does not allow the option letter to be
 100787 separated from its argument, and the options *-e*, *-i*, and *-n* require that both arguments, if
 100788 present, not be separated from the option letter.

100789 The following options shall be supported. In the following option descriptions, *column*, *lines*,
 100790 *offset*, *page*, and *width* are positive decimal integers; *gap* is a non-negative decimal integer.

- 100791 **+page** Begin output at page number *page* of the formatted input.
- 100792 **-column** Produce multi-column output that is arranged in *column* columns (the default shall
 100793 be 1) and is written down each column in the order in which the text is received
 100794 from the input file. This option should not be used with *-m*. The options *-e* and *-i*
 100795 shall be assumed for multiple text-column output. Whether or not text columns are
 100796 produced with identical vertical lengths is unspecified, but a text column shall
 100797 never exceed the length of the page (see the *-l* option). When used with *-t*, use the
 100798 minimum number of lines to write the output.
- 100799 **-a** Modify the effect of the *-column* option so that the columns are filled across the
 100800 page in a round-robin order (for example, when *column* is 2, the first input line
 100801 heads column 1, the second heads column 2, the third is the second line in column
 100802 1, and so on).
- 100803 **-d** Produce output that is double-spaced; append an extra <newline> following every
 100804 <newline> found in the input.
- 100805 **-e[*char*][*gap*]**
 100806 Expand each input <tab> to the next greater column position specified by the
 100807 formula $n*gap+1$, where *n* is an integer > 0. If *gap* is zero or is omitted, it shall
 100808 default to 8. All <tab> characters in the input shall be expanded into the
 100809 appropriate number of <space> characters. If any non-digit character, *char*, is
 100810 specified, it shall be used as the input <tab>. If the first character of the *-e* option-

Utilities

pr

100811		argument is a digit, the entire option-argument shall be assumed to be <i>gap</i> .
100812	XSI	-f Use a <form-feed> for new pages, instead of the default behavior that uses a sequence of <newline> characters. Pause before beginning the first page if the standard output is associated with a terminal.
100813		
100814		
100815		-F Use a <form-feed> for new pages, instead of the default behavior that uses a sequence of <newline> characters.
100816		
100817		-h header Use the string <i>header</i> to replace the contents of the <i>file</i> operand in the page header.
100818		-i[<i>char</i>][<i>gap</i>] In output, replace <space> characters with <tab> characters wherever one or more adjacent <space> characters reach column positions <i>gap</i> +1, 2* <i>gap</i> +1, 3* <i>gap</i> +1, and so on. If <i>gap</i> is zero or is omitted, default tab settings at every eighth column position shall be assumed. If any non-digit character, <i>char</i> , is specified, it shall be used as the output <tab>. If the first character of the -i option-argument is a digit, the entire option-argument shall be assumed to be <i>gap</i> .
100819		
100820		
100821		
100822		
100823		
100824		-l lines Override the 66-line default and reset the page length to <i>lines</i> . If <i>lines</i> is not greater than the sum of both the header and trailer depths (in lines), the <i>pr</i> utility shall suppress both the header and trailer, as if the -t option were in effect.
100825		
100826		
100827		-m Merge files. Standard output shall be formatted so the <i>pr</i> utility writes one line from each file specified by a <i>file</i> operand, side by side into text columns of equal fixed widths, in terms of the number of column positions. Implementations shall support merging of at least nine <i>file</i> operands.
100828		
100829		
100830		
100831		-n[<i>char</i>][<i>width</i>]
100832		Provide <i>width</i> -digit line numbering (default for <i>width</i> shall be 5). The number shall occupy the first <i>width</i> column positions of each text column of default output or each line of -m output. If <i>char</i> (any non-digit character) is given, it shall be appended to the line number to separate it from whatever follows (default for <i>char</i> is a <tab>).
100833		
100834		
100835		
100836		
100837		-o offset Each line of output shall be preceded by offset <space> characters. If the -o option is not specified, the default offset shall be zero. The space taken is in addition to the output line width (see the -w option below).
100838		
100839		
100840		-p Pause before beginning each page if the standard output is directed to a terminal (<i>pr</i> shall write an <alert> to standard error and wait for a <carriage-return> to be read on <i>/dev/tty</i>).
100841		
100842		
100843		-r Write no diagnostic reports on failure to open files.
100844		-s[<i>char</i>] Separate text columns by the single character <i>char</i> instead of by the appropriate number of <space> characters (default for <i>char</i> shall be <tab>).
100845		
100846		-t Write neither the five-line identifying header nor the five-line trailer usually supplied for each page. Quit writing after the last line of each file without spacing to the end of the page.
100847		
100848		
100849		-w width Set the width of the line to <i>width</i> column positions for multiple text-column output only. If the -w option is not specified and the -s option is not specified, the default width shall be 72. If the -w option is not specified and the -s option is specified, the default width shall be 512.
100850		
100851		
100852		
100853		For single column output, input lines shall not be truncated.

100854 OPERANDS

100855 The following operand shall be supported:

100856 *file* A pathname of a file to be written. If no *file* operands are specified, or if a *file*
100857 operand is ' - ', the standard input shall be used.

100858 STDIN

100859 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is ' - '.
100860 See the INPUT FILES section.

100861 INPUT FILES

100862 The input files shall be text files.

100863 The file */dev/tty* shall be used to read responses required by the **-p** option.

100864 ENVIRONMENT VARIABLES

100865 The following environment variables shall affect the execution of *pr*:

100866 *LANG* Provide a default value for the internationalization variables that are unset or null.
100867 (See XBD [Section 8.2](#) (on page 174) the precedence of internationalization variables
100868 used to determine the values of locale categories.)

100869 *LC_ALL* If set to a non-empty string value, override the values of all the other
100870 internationalization variables.

100871 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
100872 characters (for example, single-byte as opposed to multi-byte characters in
100873 arguments and input files) and which characters are defined as printable (character
100874 class **print**). Non-printable characters are still written to standard output, but are
100875 not counted for the purpose for column-width and line-length calculations.

100876 *LC_MESSAGES*

100877 Determine the locale that should be used to affect the format and contents of
100878 diagnostic messages written to standard error.

100879 *LC_TIME* Determine the format of the date and time for use in writing header lines.

100880 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

100881 *TZ* Determine the timezone used to calculate date and time strings written in header
100882 lines. If *TZ* is unset or null, an unspecified default timezone shall be used.

100883 ASYNCHRONOUS EVENTS

100884 If *pr* receives an interrupt while writing to a terminal, it shall flush all accumulated error
100885 messages to the screen before terminating.

100886 STDOUT

100887 The *pr* utility output shall be a paginated version of the original file (or files). This pagination
100888 shall be accomplished using either <form-feed> characters or a sequence of <newline>
100889 XSI characters, as controlled by the **-F** or **-f** option. Page headers shall be generated unless the **-t**
100890 option is specified. The page headers shall be of the form:

100891 "\n\n%s %s Page %d\n\n\n", <output of date>, <file>, <page number>

100892 In the POSIX locale, the <output of date> field, representing the date and time of last modification
100893 of the input file (or the current date and time if the input file is standard input), shall be
100894 equivalent to the output of the following command as it would appear if executed at the given
100895 time:

100896 date "+%b %e %H:%M %Y"

100897 without the trailing <newline>, if the page being written is from standard input. If the page
 100898 being written is not from standard input, in the POSIX locale, the same format shall be used, but
 100899 the time used shall be the modification time of the file corresponding to *file* instead of the current
 100900 time. When the *LC_TIME* locale category is not set to the POSIX locale, a different format and
 100901 order of presentation of this field may be used.

100902 If the standard input is used instead of a *file* operand, the <*file*> field shall be replaced by a null
 100903 string.

100904 If the **-h** option is specified, the <*file*> field shall be replaced by the *header* argument.

100905 **STDERR**

100906 The standard error shall be used for diagnostic messages and for alerting the terminal when **-p**
 100907 is specified.

100908 **OUTPUT FILES**

100909 None.

100910 **EXTENDED DESCRIPTION**

100911 None.

100912 **EXIT STATUS**

100913 The following exit values shall be returned:

100914 0 Successful completion.

100915 >0 An error occurred.

100916 **CONSEQUENCES OF ERRORS**

100917 Default.

100918 **APPLICATION USAGE**

100919 A conforming application must protect its first operand, if it starts with a <plus-sign>, by
 100920 preceding it with the "--" argument that denotes the end of the options. For example, *pr+x*
 100921 could be interpreted as an invalid page number or a *file* operand.

100922 **EXAMPLES**

100923 1. Print a numbered list of all files in the current directory:

100924 `ls -a | pr -n -h "Files in $(pwd) ."`

100925 2. Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

100926 `pr -3d -h "file list" file1 file2`

100927 3. Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, ...:

100928 `pr -e9 -t <file1 >file2`

100929 **RATIONALE**

100930 This utility is one of those that does not follow the Utility Syntax Guidelines because of its
 100931 historical origins. The standard developers could have added new options that obeyed the
 100932 guidelines (and marked the old options obsolescent) or devised an entirely new utility; there are
 100933 examples of both actions in this volume of POSIX.1-2008. Because of its widespread use by
 100934 historical applications, the standard developers decided to exempt this version of *pr* from many
 100935 of the guidelines.

100936 Implementations are required to accept option-arguments to the **-h**, **-l**, **-o**, and **-w** options
 100937 whether presented as part of the same argument or as a separate argument to *pr*, as suggested by
 100938 the Utility Syntax Guidelines. The **-n** and **-s** options, however, are specified as in historical

- 100939 practice because they are frequently specified without their optional arguments. If a <blank>
 100940 were allowed before the option-argument in these cases, a *file* operand could mistakenly be
 100941 interpreted as an option-argument in historical applications.
- 100942 The text about the minimum number of lines in multi-column output was included to ensure
 100943 that a best effort is made in balancing the length of the columns. There are known historical
 100944 implementations in which, for example, 60-line files are listed by *pr -2* as one column of 56 lines
 100945 and a second of 4. Although this is not a problem when a full page with headers and trailers is
 100946 produced, it would be relatively useless when used with *-t*.
- 100947 Historical implementations of the *pr* utility have differed in the action taken for the *-f* option.
 100948 BSD uses it as described here for the *-F* option; System V uses it to change trailing <newline>
 100949 characters on each page to a <form-feed> and, if standard output is a TTY device, sends an
 100950 <alert> to standard error and reads a line from */dev/tty* before the first page. There were strong
 100951 arguments from both sides of this issue concerning historical practice and as a result the *-F*
 100952 option was added. XSI-conformant systems support the System V historical actions for the *-f*
 100953 option.
- 100954 The <output of date> field in the *-l* format is specified only for the POSIX locale. As noted, the
 100955 format can be different in other locales. No mechanism for defining this is present in this volume
 100956 of POSIX.1-2008, as the appropriate vehicle is a message catalog; that is, the format should be
 100957 specified as a “message”.
- 100958 **FUTURE DIRECTIONS**
 100959 None.
- 100960 **SEE ALSO**
 100961 *expand*, *lp*
 100962 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 215)
- 100963 **CHANGE HISTORY**
 100964 First released in Issue 2.
- 100965 **Issue 6**
 100966 The following new requirements on POSIX implementations derive from alignment with the
 100967 Single UNIX Specification:
 100968 • The *-p* option is added.
- 100969 The normative text is reworded to avoid use of the term “must” for application requirements.
- 100970 **Issue 7**
 100971 PASC Interpretation 1003.2-92 #151 (SD5-XCU-ERN-44) is applied.
 100972 Austin Group Interpretation 1003.1-2001 #093 is applied.
 100973 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Utilities

printf100974 **NAME**100975 **printf** — write formatted output100976 **SYNOPSIS**100977 `printf format [argument...]`100978 **DESCRIPTION**

100979 The *printf* utility shall write formatted operands to the standard output. The *argument* operands
 100980 shall be formatted under control of the *format* operand.

100981 **OPTIONS**

100982 None.

100983 **OPERANDS**

100984 The following operands shall be supported:

100985 *format* A string describing the format to use to write the remaining operands. See the
 100986 EXTENDED DESCRIPTION section.

100987 *argument* The strings to be written to standard output, under the control of *format*. See the
 100988 EXTENDED DESCRIPTION section.

100989 **STDIN**

100990 Not used.

100991 **INPUT FILES**

100992 None.

100993 **ENVIRONMENT VARIABLES**100994 The following environment variables shall affect the execution of *printf*:

100995 *LANG* Provide a default value for the internationalization variables that are unset or null.
 100996 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables
 100997 used to determine the values of locale categories.)

100998 *LC_ALL* If set to a non-empty string value, override the values of all the other
 100999 internationalization variables.

101000 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 101001 characters (for example, single-byte as opposed to multi-byte characters in
 101002 arguments).

101003 *LC_MESSAGES* Determine the locale that should be used to affect the format and contents of
 101004 diagnostic messages written to standard error.
 101005

101006 *LC_NUMERIC* Determine the locale for numeric formatting. It shall affect the format of numbers
 101007 written using the *e*, *E*, *f*, *g*, and *G* conversion specifier characters (if supported).
 101008

101009 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

101010 **ASYNCHRONOUS EVENTS**

101011 Default.

101012 **STDOUT**

101013 See the EXTENDED DESCRIPTION section.

printf*Utilities*101014 **STDERR**

101015 The standard error shall be used only for diagnostic messages.

101016 **OUTPUT FILES**

101017 None.

101018 **EXTENDED DESCRIPTION**

101019 The *format* operand shall be used as the *format* string described in XBD Chapter 5 (on page 121)
101020 with the following exceptions:

- 101021 1. A <space> in the format string, in any context other than a flag of a conversion
101022 specification, shall be treated as an ordinary character that is copied to the output.
- 101023 2. A 'Δ' character in the format string shall be treated as a 'Δ' character, not as a <space>.
- 101024 3. In addition to the escape sequences shown in XBD Chapter 5 (on page 121) ('\\', '\a',
101025 '\b', '\f', '\n', '\r', '\t', '\v'), "\ddd", where *ddd* is a one, two, or three-digit
101026 octal number, shall be written as a byte with the numeric value specified by the octal
101027 number.
- 101028 4. The implementation shall not precede or follow output from the *d* or *u* conversion
101029 specifiers with <blank> characters not specified by the *format* operand.
- 101030 5. The implementation shall not precede output from the *o* conversion specifier with zeros
101031 not specified by the *format* operand.
- 101032 6. The *a*, *A*, *e*, *E*, *f*, *F*, *g*, and *G* conversion specifiers need not be supported.
- 101033 7. An additional conversion specifier character, *b*, shall be supported as follows. The
101034 argument shall be taken to be a string that may contain <backslash>-escape sequences.
101035 The following <backslash>-escape sequences shall be supported:
 - 101036 — The escape sequences listed in XBD Chapter 5 (on page 121) ('\\', '\a', '\b',
101037 '\f', '\n', '\r', '\t', '\v'), which shall be converted to the characters they
101038 represent
 - 101039 — "\0ddd", where *ddd* is a zero, one, two, or three-digit octal number that shall be
101040 converted to a byte with the numeric value specified by the octal number
 - 101041 — '\c', which shall not be written and shall cause *printf* to ignore any remaining
101042 characters in the string operand containing it, any remaining string operands, and
101043 any additional characters in the *format* operand
- 101044 The interpretation of a <backslash> followed by any other sequence of characters is
101045 unspecified.
- 101046 Bytes from the converted string shall be written until the end of the string or the number
101047 of bytes indicated by the precision specification is reached. If the precision is omitted, it
101048 shall be taken to be infinite, so all bytes up to the end of the converted string shall be
101049 written.
- 101050 8. For each conversion specification that consumes an argument, the next argument operand
101051 shall be evaluated and converted to the appropriate type for the conversion as specified
101052 below.
- 101053 9. The *format* operand shall be reused as often as necessary to satisfy the argument
101054 operands. Any extra *c* or *s* conversion specifiers shall be evaluated as if a null string
101055 argument were supplied; other extra conversion specifications shall be evaluated as if a
101056 zero argument were supplied. If the *format* operand contains no conversion specifications
101057 and *argument* operands are present, the results are unspecified.

10. If a character sequence in the *format* operand begins with a '%' character, but does not form a valid conversion specification, the behavior is unspecified.

11. The argument to the *c* conversion specifier can be a string containing zero or more bytes. If it contains one or more bytes, the first byte shall be written and any additional bytes shall be ignored. If the argument is an empty string, it is unspecified whether nothing is written or a null byte is written.

The *argument* operands shall be treated as strings if the corresponding conversion specifier is *b*, *c*, or *s*, and shall be evaluated as if by the *strtod()* function if the corresponding conversion specifier is *a*, *A*, *e*, *E*, *f*, *F*, *g*, or *G*. Otherwise, they shall be evaluated as unsuffixed C integer constants, as described by the ISO C standard, with the following extensions:

- A leading <plus-sign> or minus-sign shall be allowed.
- If the leading character is a single-quote or double-quote, the value shall be the numeric value in the underlying codeset of the character following the single-quote or double-quote.
- Suffixed integer constants may be allowed.

If an argument operand cannot be completely converted into an internal value appropriate to the corresponding conversion specification, a diagnostic message shall be written to standard error and the utility shall not exit with a zero exit status, but shall continue processing any remaining operands and shall write the value accumulated at the time the error was detected to standard output.

It is not considered an error if an argument operand is not completely used for a *c* or *s* conversion.

EXIT STATUS

The following exit values shall be returned:

- 0 Successful completion.
- >0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The floating-point formatting conversion specifications of *printf()* are not required because all arithmetic in the shell is integer arithmetic. The *awk* utility performs floating-point calculations and provides its own **printf** function. The *bc* utility can perform arbitrary-precision floating-point arithmetic, but does not provide extensive formatting capabilities. (This *printf* utility cannot really be used to format *bc* output; it does not support arbitrary precision.) Implementations are encouraged to support the floating-point conversions as an extension.

Note that this *printf* utility, like the *printf()* function defined in the System Interfaces volume of POSIX.1-2008 on which it is based, makes no special provision for dealing with multi-byte characters when using the *%c* conversion specification or when a precision is specified in a *%b* or *%s* conversion specification. Applications should be extremely cautious using either of these features when there are multi-byte characters in the character set.

No provision is made in this volume of POSIX.1-2008 which allows field widths and precisions to be specified as '*' since the '*' can be replaced directly in the *format* operand using shell variable substitution. Implementations can also provide this feature as an extension if they so choose.

printf*Utilities*

Hexadecimal character constants as defined in the ISO C standard are not recognized in the *format* operand because there is no consistent way to detect the end of the constant. Octal character constants are limited to, at most, three octal digits, but hexadecimal character constants are only terminated by a non-hex-digit character. In the ISO C standard, the "##" concatenation operator can be used to terminate a constant and follow it with a hexadecimal character to be written. In the shell, concatenation occurs before the *printf* utility has a chance to parse the end of the hexadecimal constant.

The %b conversion specification is not part of the ISO C standard; it has been added here as a portable way to process <backslash>-escapes expanded in string operands as provided by the *echo* utility. See also the APPLICATION USAGE section of *echo* (on page 2615) for ways to use *printf* as a replacement for all of the traditional versions of the *echo* utility.

If an argument cannot be parsed correctly for the corresponding conversion specification, the *printf* utility is required to report an error. Thus, overflow and extraneous characters at the end of an argument being used for a numeric conversion shall be reported as errors.

EXAMPLES

To alert the user and then print and read a series of prompts:

```
printf "\aPlease fill in the following: \nName: "
read name
printf "Phone number: "
read phone
```

To read out a list of right and wrong answers from a file, calculate the percentage correctly, and print them out. The numbers are right-justified and separated by a single <tab>. The percentage is written to one decimal place of accuracy:

```
while read right wrong ; do
    percent=$(echo "scale=1;($right*100)/($right+$wrong)" | bc)
    printf "%2d right\t%2d wrong\t(%%)\n" \
        $right $wrong $percent
done < database_file
```

The command:

```
printf "%5d%4d\n". 1 21 321 4321 54321
```

produces:

```
    1   21
   321 4321
  54321  0
```

Note that the *format* operand is used three times to print all of the given strings and that a '0' was supplied by *printf* to satisfy the last %4d conversion specification.

The *printf* utility is required to notify the user when conversion errors are detected while producing numeric output; thus, the following results would be expected on an implementation with 32-bit twos-complement integers when %d is specified as the *format* operand:

Argument	Standard Output	Diagnostic Output
5a	5	printf: "5a" not completely converted
9999999999	2147483647	printf: "9999999999" arithmetic overflow
-9999999999	-2147483648	printf: "-9999999999" arithmetic overflow
ABC	0	printf: "ABC" expected numeric value

The diagnostic message format is not specified, but these examples convey the type of information that should be reported. Note that the value shown on standard output is what would be expected as the return value from the *strtol()* function as defined in the System Interfaces volume of POSIX.1-2008. A similar correspondence exists between %u and *strtoul()* and %e, %f, and %g (if the implementation supports floating-point conversions) and *strtod()*.

In a locale using the ISO/IEC 646:1991 standard as the underlying codeset, the command:

```
printf "%d\n" 3 +3 -3 \'3 \"+3 "'-3"
```

produces:

3 Numeric value of constant 3

3 Numeric value of constant 3

-3 Numeric value of constant -3

51 Numeric value of the character '3' in the ISO/IEC 646:1991 standard codeset

43 Numeric value of the character '+' in the ISO/IEC 646:1991 standard codeset

45 Numeric value of the character '-' in the ISO/IEC 646:1991 standard codeset

Note that in a locale with multi-byte characters, the value of a character is intended to be the value of the equivalent of the **wchar_t** representation of the character as described in the System Interfaces volume of POSIX.1-2008.

RATIONALE

The *printf* utility was added to provide functionality that has historically been provided by *echo*. However, due to irreconcilable differences in the various versions of *echo* extant, the version has few special features, leaving those to this new *printf* utility, which is based on one in the Ninth Edition system.

The EXTENDED DESCRIPTION section almost exactly matches the *printf()* function in the ISO C standard, although it is described in terms of the file format notation in XBD Chapter 5 (on page 121).

Earlier versions of this standard specified that arguments for all conversions other than b, c, and s were evaluated in the same way (as C constants, but with stated exceptions). For implementations supporting the floating-point conversions it was not clear whether integer conversions need only accept integer constants and floating-point conversions need only accept floating-point constants, or whether both types of conversions should accept both types of constants. Also by not distinguishing between them, the requirement relating to a leading single-quote or double-quote applied to floating-point conversions even though this provided no useful functionality to applications that was not already available through the integer conversions. The current standard clarifies the situation by specifying that the arguments for floating-point conversions are evaluated as if by *strtod()*, and the arguments for integer conversions are evaluated as C integer constants, with the special treatment of leading single-quote and double-quote applying only to integer conversions.

printf*Utilities*101184 **FUTURE DIRECTIONS**

101185 None.

101186 **SEE ALSO**101187 *awk, bc, echo*101188 XBD [Chapter 5](#) (on page 121), [Chapter 8](#) (on page 173)101189 XSH *fprintf()*, *strtod()*101190 **CHANGE HISTORY**

101191 First released in Issue 4.

101192 **Issue 7**

101193 Austin Group Interpretations 1003.1-2001 #175 and #177 are applied.

101194 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

IECNORM.COM : Click to view the full PDF of ISO/IEC/IEEE 9945:2009

101195 **NAME**101196 *prs* — print an SCCS file (**DEVELOPMENT**)101197 **SYNOPSIS**

```

101198 XSI      prs [-a] [-d dataspec] [-r[SID]] file...
101199          prs [-e|-l] -c cutoff [-d dataspec] file...
101200          prs [-e|-l] -r[SID] [-d dataspec] file...

```

101201 **DESCRIPTION**

101202 The *prs* utility shall write to standard output parts or all of an SCCS file in a user-supplied
 101203 format.

101204 **OPTIONS**

101205 The *prs* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the *-r* option has an
 101206 optional option-argument. This optional option-argument cannot be presented as a separate
 101207 argument. The following options shall be supported:

101208 *-d dataspec* Specify the output data specification. The *dataspec* shall be a string consisting of
 101209 SCCS file *data keywords* (see [Data Keywords](#), on page 3056) interspersed with
 101210 optional user-supplied text.

101211 *-r[SID]* Specify the SCCS identification string (*SID*) of a delta for which information is
 101212 desired. If no *SID* option-argument is specified, the *SID* of the most recently
 101213 created delta shall be assumed.

101214 *-e* Request information for all deltas created earlier than and including the delta
 101215 designated via the *-r* option or the date-time given by the *-c* option.

101216 *-l* Request information for all deltas created later than and including the delta
 101217 designated via the *-r* option or the date-time given by the *-c* option.

101218 *-c cutoff* Indicate the *cutoff* date-time, in the form:

101219 *YY[MM[DD[HH[MM[SS]]]]]*

101220 For the *YY* component, values in the range [69,99] shall refer to years 1969 to 1999
 101221 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.

101222 **Note:** It is expected that in a future version of this standard the default century inferred
 101223 from a 2-digit year will change. (This would apply to all commands accepting a
 101224 2-digit year as input.)

101225 No changes (deltas) to the SCCS file that were created after the specified *cutoff*
 101226 date-time shall be included in the output. Units omitted from the date-time default
 101227 to their maximum possible values; for example, *-c 7502* is equivalent to
 101228 *-c 750228235959*.

101229 *-a* Request writing of information for both removed—that is, *delta type=R* (see
 101230 *rmDEL*)—and existing—that is, *delta type=D*,—deltas. If the *-a* option is not
 101231 specified, information for existing deltas only shall be provided.

101232 **OPERANDS**

101233 The following operand shall be supported:

101234 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *prs*
 101235 utility shall behave as though each file in the directory were specified as a named
 101236 file, except that non-SCCS files (last component of the pathname does not begin
 101237 with *s.*) and unreadable files shall be silently ignored.

101238 If exactly one *file* operand appears, and it is '–', the standard input shall be read;
 101239 each line of the standard input shall be taken to be the name of an SCCS file to be
 101240 processed. Non-SCCS files and unreadable files shall be silently ignored.

101241 STDIN

101242 The standard input shall be a text file used only when the *file* operand is specified as '–'. Each
 101243 line of the text file shall be interpreted as an SCCS pathname.

101244 INPUT FILES

101245 Any SCCS files displayed are files of an unspecified format.

101246 ENVIRONMENT VARIABLES

101247 The following environment variables shall affect the execution of *prs*:

101248 *LANG* Provide a default value for the internationalization variables that are unset or null.
 101249 (See XBD [Section 8.2](#) (on page 174) the precedence of internationalization variables
 101250 used to determine the values of locale categories.)

101251 *LC_ALL* If set to a non-empty string value, override the values of all the other
 101252 internationalization variables.

101253 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 101254 characters (for example, single-byte as opposed to multi-byte characters in
 101255 arguments and input files).

101256 *LC_MESSAGES*

101257 Determine the locale that should be used to affect the format and contents of
 101258 diagnostic messages written to standard error.

101259 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

101260 ASYNCHRONOUS EVENTS

101261 Default.

101262 STDOUT

101263 The standard output shall be a text file whose format is dependent on the data keywords
 101264 specified with the *–d* option.

101265 Data Keywords

101266 Data keywords specify which parts of an SCCS file shall be retrieved and output. All parts of an
 101267 SCCS file have an associated data keyword. A data keyword may appear in a *dataspec* multiple
 101268 times.

101269 The information written by *prs* shall consist of:

- 101270 1. The user-supplied text
- 101271 2. Appropriate values (extracted from the SCCS file) substituted for the recognized data
 101272 keywords in the order of appearance in the *dataspec*

101273 The format of a data keyword value shall either be simple ('S'), in which keyword substitution
 101274 is direct, or multi-line ('M').

101275 User-supplied text shall be any text other than recognized data keywords. A <tab> shall be
 101276 specified by '\t' and <newline> by '\n'. When the *–r* option is not specified, the default
 101277 *dataspec* shall be:

101278 :PN: :\n\n

101279 and the following *dataspec* shall be used for each selected delta: