INTERNATIONAL STANDARD

ISO/IEC 13210

ANSI/IEEE Std 1003.3

First edition
1994-12-30

Information technology — Test methods for measuring conformance to POSIX

Technologies de l'information — Méthodes d'essai pour mesurer la conformité à POSIX

Reference number
ISO/IEC 13210:1994(E)
ANSI/IEEE
Std 1003.3-1991 edition

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

ISBN 1-55937-494-2

December 30, 1994 *SH94251*

International Standard ISO/IEC 13210: 1994 (E)
ANSI/IEEE Std 1003.3-1991

Information technology—Test methods for measuring conformance to POSIX

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

Approved March 21, 1991

IEEE Standards Board

Approved August 9, 1991

American National Standards Institute

Approved 1994 by the

International Organization for Standardization
and by the
International Electrotechnical Commission

Abstract: The general requirements and test methods for measuring conformance to POSIX standards are defined. This document is aimed primarily at working groups developing test methods for POSIX standards, developers of POSIX test methods, and users of POSIX test methods.

Keywords: assertion, assertion test, base assertion, conditional feature, extended assertions, POSIX, POSIX Conformance Document, POSIX Conformance Test Procedure, POSIX Conformance Test Suite, test method, test result code

Adopted as an International Standard by the
International Organization for Standardization
and by the
International Electrotechnical Commission

Published by
The Institute of Electrical and Electronics Engineers, Inc.

Foreword

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

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 13210 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

Annexes A and B form an integral part of this International Standard.

International Organization for Standardization/International Electrotechnical Commission
Case postale 56 • CH-1211 Genève 20 • Switzerland

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

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

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

IEEE standards documents may involve the use of patented technology. Their approval by the Institute of Electrical and Electronics Engineers, Inc. does not mean that using such technology for the purpose of conforming to such standards is authorized by the patent owner. It is the obligation of the user of such technology to obtain all necessary permissions.

Contents

FIGURES

TABLES

Introduction

(This Introduction is not a normative part of ISO/IEC 13210 Information Technology— Test methods for measuring conformance to POSIX, but is included for information only.)

This standard defines the general requirements and test methods for measuring conformance to POSIX standards. This document is aimed primarily at working groups developing test methods for POSIX standards, developers of POSIX test methods, and users of POSIX test methods.

Organization of the Standard

This document is organized into sections and clauses. Some of these, such as 1.1, are mandated by the IEEE and other standards bodies.

The sections are:

(1) General

(2) Terminology and General Requirements

(3) Test Methods Conformance to This Standard

(4) Testing Levels and Complexity Levels

(5) Classification of Assertions

(6) Writing Assertions

(7) Assertion Test Output

(8) Statement of Conformance to a POSIX Standard

— A *Topical Index* points to the definition and/or usage of key words and phrases.

The foreword, introduction, any footnotes, and the informative annexes are not considered part of this International Standard.

Revisions and Supplements to This Standard

No activities to extend this International Standard to address additional requirements are in progress. Extension efforts may be anticipated in the future. This is an outline of how extensions can be incorporated.

Extensions are approved as "amendments" or "revisions" to this document, following the IEEE Standards Procedures.

Approved amendments are published separately until the full document is reprinted and such amendments are incorporated in their proper positions.

If you have any question about the completeness of your version, you may contact the IEEE Standards Office [+1 (908) 562-3800] to determine what supplements have been published.

If you have interest in participating in the TCOS working groups addressing these issues, please send your name, address, and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, USA, and ask to have this forwarded to the chairperson of the appropriate TCOS working group.

IEEE Std 1003.3-1991 was prepared by the 1003.3 Working Group, sponsored by the Technical Committee on Operating Systems and Application Environments of the IEEE Computer Society. At the time this standard was approved, the membership of the 1003.3 Working Group was as follows:

Technical Committee on Operating Systems and Application Environments (TCOS)

Chair: Jehan-François Pâris

Standards Subcommittee for TCOS

Chair:	Jim Isaak
Treasurer:	Quin Hahn
Secretary:	Shane McCarron

1003.3 Working Group Officials

Chair:	Roger Martin
Vice Chair:	N. Ray Wilkes (1989-1990)
	Carol Raye (1987-1988)
Editor:	Anthony Cincotta
Secretary:	Lowell Johnson (1989-1990)
	Doris Lebovits (1987-1988)

Sanjay Agraharam*	Lorraine Kevra	Fred Noz
Jim Bound	Martin J. Kirk	Bob Palm
Ken Gibb	Mark Lamonds*	Gerald Powell*
Judy Guist	Robert M. Lenk	Ravi Tavakley*
Rich Hamm	Kevin Lewis	Donn S. Terry
Steve Henderson*	Marty McGowan	Andrew Twigger*
Scott Jameson	Jim Moe*	Bruce Weiner*
Greg Jones	Anita Mundkur*	Brian Weis

In the preceding list, those individuals identified with asterisks (*) served during the balloting period as Technical Reviewers for resolving comments and objections to designated portions of the standard.

The following persons were members of the 1003.3 Balloting Group that approved the standard for submission to the IEEE Standards Board:

Open Software Foundation	David Chen	Institutional Representative
X/Open Co. Ltd	Mike Lambert	Institutional Representative
David Athersych	Charles E. Hammons	Barry Needham
Geoff Baldwin	Allen Hankinson	Fred Noz
Jerome E. Banasik	Craig Harmer	Alan F. Nugent
Robert Barned	Dale Harris	John C. Penney
Kabekode V. S. Bhat	John L. Hill	P. J. Plauger
Robert Bismuth	David F. Hinnant	Gerald Powell
James Bohem	Lee A. Hollaar	Scott E. Preece
Robert Borochoff	Terrence W. Holm	James M. Purtilo
Keith Bostic	Jim Isaak	Carol Raye
James P. Bound	Dan Iuster	Christopher J. Riddick
Phyllis Eve Bregman	Hal Jespersen	R. Hughes Rowlands
A. Winsor Brown	Lowell G. Johnson	Robert Sarr
F. Lee Brown, Jr.	Greg Jones	Norman Schneidewind
Nicholas A. Camillone	Lorraine C. Kevra	Leonard W. Seagren
Clyde Camp	Jeff Kimmel	Karen Sheaffer
John Caywood	M. J. Kirk	Dan Shia
Kilnam Chon	Dale L. Kirkland	Mukesh Singhal
Chan F. Chong	Kenneth C. Klingman	Richard Sniderman
Anthony V. Cincotta	D. Richard Kuhn	Douglas H. Steves
Robert L. Claeson	Robin B. Lake	Scott A. Sutter
Richard Cornelius	Mark Lamonds	Robert Switzer
William M. Corwin	Doris Lebovits	Ravi Tavakley
William Cox	Robert M. Lenk	Donn Terry
Donald W. Cragun	David Lennert	Gary F. Tom
Ava Maria De Alvare	Kevin Lewis	A. T. Twigger
Terence Dowling	Joseph F. P. Luhukay	Mark-Rene Uchida
Stephen A. Dum	Robert J. Makowski	Michael W. Vannier
John D. Earls	Roger J. Martin	John W. Walz
Ron Elliott	Joberto S. B. Martins	Alan G. Weaver
Philip H. Enslow	Shane McCarron	Bruce Weiner
Ken Faubel	Martin J. McGowan III	Brian Weis
Terence Fong	Robert W. McWhirter	Peter J. Weyman
Kenneth R. Gibb	Paul Merry	Andrew E. Wheeler
Michel Gien	Doug Michels	Nicholas Wilkes
Gregory W. Goddard	James M. Moe	Oren Yuen
Dave Grinderland	Anita Mundkur	Alaa Zeineldine
Judy Guist	Martha Nalebuff	Jason Zions

When the IEEE Standards Board approved this standard on March 21, 1991, it had the following membership:

Marco W. Migliaro, *Chair* **Donald C. Loughry,** *Vice Chair*
Andrew G. Salem, *Secretary*

Dennis Bodson
Paul L. Borrill
Clyde Camp
James M. Daly
Donald C. Fleckenstein
Jay Forster*
David F. Franklin
Ingrid Fromm
Thomas L. Hannan
Donald N. Heirman
Kenneth D. Hendrix
John W. Horch
Ben C. Johnson
Ivor N. Knight
Joseph L. Koepfinger*
Irving Kolodny
Michael A. Lawler
John E. May, Jr.
Lawrence V. McCall
Donald T. Michael*
Stig L. Nilsson
John L. Rankine
Ronald H. Reimer
Gary S. Robinson
Terrance R. Whittemore

*Member Emeritus

Acknowledgments

We wish to thank the National Institute of Standards and Technology for donating significant computer, printing, and editing resources to the production of this standard.

Also we wish to thank the organizations employing the members of the Working Group and the Balloting Group for both covering the expenses related to attending and participating in meetings and for donating the time required both in and out of meetings for this effort.

This page intentionally left blank

Information Technology— Test methods for measuring conformance to POSIX

Section 1: General

1.1 Scope

This International Standard is applicable to the development and use of conformance testing methodologies for POSIX standards. The generic test methods identified in this International Standard shall be used in conjunction with test methods identified for a specific standard.

This International Standard is intended for use by working groups developing test methods for POSIX standards, developers of POSIX test methods, and users of POSIX test methods.

The purpose of this International Standard is to define general rules for developing test assertions and related test methods for measuring conformance of an implementation to POSIX standards. Test methods may include POSIX Conformance Test Suites (PCTS), POSIX Conformance Test Procedures (PCTP), and audits of POSIX Conformance Documents (PCD).

Testing conformance of an implementation to a standard includes testing the claimed capabilities and behavior of the implementation with respect to the conformance requirements of the standard. Test methods are intended to provide a reasonable, practical assurance that the implementation conforms to the standard. Use of these test methods will not guarantee conformance of an implementation to the standard; that normally would require exhaustive testing (see 4.2.1), which is impractical for both technical and economic reasons.

1.2 Normative References

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

{1} IEEE Std 729-1983, *IEEE Glossary of Software Engineering Terminology (ANSI).*

{2} ISO/IEC 9945-1: 1990 (ISO/IEC 9945-1: 1990), *Information technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].*

Section 2: Terminology and General Requirements

2.1 Conventions

This International Standard uses the following typographic conventions:

(1) The *italic* font is used for:

— The initial appearances of defined terms

— Cross-references to defined terms within 2.2.1, 2.2.2, and 2.2.3

— Parameters (option arguments and operands) that are generally substituted with real values by the application

— C language data types and function names

— Global external variable names

(2) The **bold** font is used for:

— Test result codes

— Assertion type classification

(3) The `constant-width` (Courier) font is used:

— To illustrate examples of system input or output where exact usage is depicted

— For references to utility names and C language headers

(4) Symbolic *errno* names returned by functions are represented as [symbolic_name].

(5) Symbolic constant limits are represented as {symbolic_constant_limit}.

(6) Symbolic constant options are represented as {Option_name}.

(7) Notes provided as parts of labeled tables and figures are integral parts of this International Standard (normative). Footnotes and Notes within the body of the text are for information only (informative).

(8) Defined names that are usually in lowercase, particularly function names, are never used at the beginning of a sentence or anywhere else where regular English usage would require them to be capitalized.

(9) Mathematical symbols such as <, ≤, etc. are only used in formulas, assertion classification assignments, and conditional clauses preceding conditional assertions.

(10) In some cases tabular information is presented "inline"; in others it is presented in a separately labeled table. This arrangement was employed purely for ease of typesetting and there is no normative difference between these two cases.

(11) The conventions listed above are for ease of reading only. Editorial inconsistencies in the use of typography are unintentional and have no normative meaning in this International Standard.

A summary of typographical conventions is shown in Table 2-1.

Table 2-1 – Typographical Conventions

Reference	Example
C Language Header	`<sys/stat.h>`
Command Name	`awk`
Command Option	`-c`
Command Option With Argument	`-w` *width*
Data Types	*long*
Defined Terms	*file*
Environment Variables	**PATH**
Error Number	[EINTR]
File Name	`filename`
Function Argument Declaration	extern unsigned long int
Function Argument	*arg1*
Function Declaration	`int fstat(int` *fildes*`, struct stat *`*buf*`);`
Function Name	*funct*()
Global External	*errno*
Implementation-dependent Limit	{MAX_INPUT}
Metavariable	<*>
Operand	*file_name*
Output	**foo**
Parameters	<*directory pathname*>
Section Reference	(see Section 1)
Clause Reference	(see 1.m)
Subclause Reference	(see 1.n.m...)
Special Character	`<newline>`
Symbolic Constant Limit	{LINK_MAX}
Symbolic Constant Option	{_POSIX_JOB_CONTROL}
Table Reference	Table 6
Variable	*st_atime*

2.2 Definitions

2.2.1 Terminology

For the purposes of this International Standard, the following definitions apply:

2.2.1.1 may: an option for test methods.

In this International Standard, *need not* is used as the negative of *may*.

2.2.1.2 shall: A requirement for test methods.

2.2.1.3 should: A recommended practice for test methods.

2.2.2 General Terms

For the purposes of this International Standard, the following definitions apply:

2.2.2.1 assertion: A statement of functionality or behavior for a POSIX element that is derived from the POSIX standard being tested and that is true for a conforming POSIX implementation.

2.2.2.2 assertion number: The numeric identifier assigned to an assertion.

The name of the element and the assertion number together uniquely identify an assertion.

2.2.2.3 assertion test: The software or procedural methods that ascertain the conformance of a POSIX implementation to an assertion.

2.2.2.4 base assertion: An assertion that is required to be tested for required features and for implemented conditional features.

2.2.2.5 conditional feature: A feature or behavior referred to in a POSIX standard that need not be present on all conforming implementations.

2.2.2.6 development system: The computer system used to compile and configure a PCTS.

2.2.2.7 element: A functional interface or a namespace allocation.

Examples of elements are C functions or utility programs. Examples of namespace allocation include headers or error return value constants.

2.2.2.8 extended assertion: An assertion that is not required to be tested.

2.2.2.9 POSIX Conformance Document (PCD): The conformance document required by a POSIX standard.

2.2.2.10 POSIX Conformance Test Procedure (PCTP): The nonsoftware procedures possibly used in conjunction with other test methods to measure conformance.

2.2.2.11 POSIX Conformance Test Suite (PCTS): The collection of software possibly used in conjunction with other test methods to measure conformance.

2.2.2.12 required feature: Either a single facility or behavior, or one of a pair of alternative facilities or behaviors, required by a POSIX standard that is always present on a conforming implementation.

2.2.2.13 target system: The combination of the computer system on which the PCTS is executed and the parts of the development system that are used to generate the executable code of a PCTS.

2.2.2.14 test method: The software, procedures, or other means specified by a POSIX standard to measure conformance.

Test methods may include a PCTS, PCTP, or an audit of a PCD.

2.2.2.15 test result code: A value that describes the result of an assertion test.

2.2.3 Abbreviations

For the purposes of this International Standard, the following abbreviations apply:

2.2.3.1 IEEE: The Institute of Electrical and Electronics Engineers.

2.2.3.2 PCD: POSIX Conformance Document.

2.2.3.3 PCTP: POSIX Conformance Test Procedures.

2.2.3.4 PCTS: POSIX Conformance Test Suite.

2.2.3.5 POSIX:[1)] Colloquial name for the collection of IEEE 1003 standards, draft standards, and projects.

2.2.3.6 POSIX.n: Term used to refer to a specific IEEE P1003.n project or its products. For example, POSIX.1 represents the P1003.1 project and its associated products, IEEE Std 1003.1-1990, ISO/IEC 9945-1: 1990, and IEEE Std 1003.1c-1993.

2.2.3.7 POSIX.3: This standard.

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

Section 3: Test Methods Conformance to This Standard

3.1 Conformance Criteria

Test methods that conform to POSIX.3 shall meet all of the following criteria:

- The test methods shall document the POSIX.n test method specifications to which they conform.
- The test methods shall test base assertions for required features and implemented conditional features of the POSIX standard for which conformance is being measured. If an assertion lists a set of instances (often separated by the word *or*), the assertion test shall test each specified instance.
- The PCTS shall consist of automated assertion tests to the extent possible.
- The documentation of the PCTS shall include all the information necessary to install, configure, and execute the PCTS.
- The documentation of the test methods shall include, if needed, instructions for performing the PCTP and an audit of the PCD.
- The documentation of the test methods shall describe how to gather and interpret the results.
- Additional criteria shall be specified in applicable POSIX.n standards.

This page intentionally left blank

Section 4: Testing Levels and Complexity Levels

4.1 Introduction

In principle, the objective of conformance testing is to establish whether the implementation being tested conforms to the specification in the relevant standard. Practical limitations make it impossible to be exhaustive, and economic considerations may restrict testing still further.

Due to these limitations, during the design and development of test methods the complexity level of an element will dictate the level of testing that satisfies conformance requirements.

Therefore, this International Standard distinguishes three major levels of testing, according to the extent to which they provide an indication of conformance, and three major levels of element complexity.

The three major levels of testing are:

— Exhaustive Testing

— Thorough Testing

— Identification Testing

The three major levels of element complexity are:

— Simple

— Intermediate

— Complex

4.2 Testing Levels

4.2.1 Exhaustive Testing

Exhaustive testing seeks to verify the behavior of every aspect of an element, including all permutations. For example, exhaustive testing of a given user command would require testing the command with no options, with each option, with each pair of options, and so on, up to every permutation of options.

The various command options and permutations rapidly approach numbers too large to reach execution completion in any realistic time frame. As an example, there are approximately 37 unique error conditions in POSIX.1. The occurrence of one error can (and often does) affect the proper detection of another error. An exhaustive test of the 37 errors would require not just one test per error, but one

test per possible permutation of errors. Thus, instead of 37 tests, billions of tests would be needed (2 to the 37th power).

Exhaustive testing is normally infeasible.

4.2.2 Thorough Testing

Thorough testing is a useful alternative to exhaustive testing. Thorough testing seeks to verify the behavior of every aspect of an element, but does not include all permutations. For example, to perform thorough testing of a given command, the command shall be tested with no options, then with each option individually. Possible combinations of options may also be tested.

Given the above example concerning the 37 error conditions, thorough testing would require 38 tests, a much more manageable number.

Thorough testing is a feasible solution. During testing of a multitude of individual elements, certain combinations of subelements are coincidentally tested.

In this discussion, it is helpful to consider the number of assertions that can be derived from the specification of each software element. Thorough testing aims to verify each individual aspect in isolation and is thus much more feasible than exhaustive testing. However, even thorough testing approaches infeasibility when the sheer number of aspects of an element is very large. Therefore, a third level of testing is defined.

4.2.3 Identification Testing

Identification testing seeks to verify some distinguishing characteristic of the element in question. It consists of a cursory examination of the element, invoking it with the minimal command syntax and verifying its minimal function.

For example, identification testing of a C compiler would distinguish it from a compiler for another language on the system, but not necessarily from any other C compiler on this or another system. It would not require a verification of all the syntax or functions specified in the C compiler manual. Proper identification testing of a C compiler would be to verify the minimal program constructs necessary to establish its distinguishing characteristic as a C compiler.

4.3 Complexity Levels

4.3.1 Simple

Simple elements are those that are wholly defined in the description for that element. Simple elements are those that have a few assertions to test and whose functionality does not depend on other elements defined within the POSIX standard in which they are defined. Examples of simple elements include the `cat` utility specified in POSIX.2 or the *close*() function specified in POSIX.1. Simple elements should be thoroughly tested.

4.3.2 Intermediate

Intermediate elements are those that have a moderate number of assertions and may depend upon the functionality of other elements defined within the POSIX standard in which they are defined. Examples of intermediate elements are the `grep` and `sed` utilities specified in POSIX.2, which support their own functionality in addition to regular expressions. Thorough testing should be a goal for intermediate elements, but it may not be feasible in some cases.

4.3.3 Complex

Complex elements are those that implement a language, depend upon the function of intermediate elements, or have effects on hardware devices. Typically, complex elements need a large number of assertion tests to test them thoroughly. Examples of complex elements are the `sh` and `awk` utilities as specified in POSIX.2, which implement a language in addition to regular expressions. For complex elements, thorough testing may be limited to specific areas of the element.

4.4 Conclusion

It follows from the definitions of testing levels and of element complexity levels that the functionality of each element must be analyzed and evaluated. Since the number of possible combinations of options, events, and timing of events is unreasonably large, testing normally cannot be exhaustive.

It follows from the informal nature of the definition of thorough testing that thoroughness of testing will vary from one PCTS to another and, within a PCTS, from one assertion to another. These are choices the PCTS implementor must make. It is not within the scope of this International Standard to define the recommended testing level so precisely as to preclude this degree of freedom.

This page intentionally left blank

Section 5: Classification of Assertions

5.1 Method of Classification

Each assertion falls into one of four categories represented by the two-by-two matrix as shown in Table 5-1.

Table 5-1 – Classification of Assertions

	Base Assertion	Extended Assertion
Required Feature	A	B
Conditional Feature	C	D

The rows of the assertion matrix correspond to whether the POSIX feature is required or conditional (see 2.2.2). The columns of the assertion matrix correspond to whether or not a test for the assertion is required. Those assertions not required to be tested are known as extended assertions. Whenever possible, the development of assertion tests for extended assertions is encouraged.

In some cases it may be appropriate to develop an assertion test that partially, but not fully, tests an extended assertion. For example, it is sometimes possible to detect failure to conform in certain instances, but not possible to detect all failures. Any assertion test that detects such failures shall produce a test result code of **FAIL** (see Section 7) for that assertion. Any assertion test that does not adequately test the assertion and that completes correctly on the system under test without detecting a failure in that system shall return a test result code of **UNTESTED.** The following is a list of reasons to classify assertions as extended:

Reason: 1 There is no known portable test method for this assertion.

Reason: 2 The corresponding statement in the POSIX standard to which conformance is being measured is not specific enough to write a portable test.

Reason: 3 There is no known reliable test method for this assertion.

Reason: 4 The assertion test requires setup procedures that involve an unreasonable amount of effort by the user of a test method.

Reason: 5 The assertion test would require an unreasonable amount of time or resources on most systems.

Reason: 6 Creating an assertion test would require an unreasonable amount of test development time.

Reason: 7 The assertion test could have an adverse effect on completion of a test method.

Those assertions that are not extended are known as base assertions. Base assertions for required features and implemented conditional features shall be tested.

Thus, each assertion falls into one of four categories:

(A) Base assertion of a required feature.

(B) Extended assertion of a required feature.

(C) Base assertion of a conditional feature.

(D) Extended assertion of a conditional feature.

Section 6: Writing Assertions

This section describes how test assertions in POSIX standards shall be written.

6.1 Assertion Determination

An assertion is written for each definitive statement in a POSIX standard that pertains to an implementation. A corresponding assertion test is designed to determine whether the statement is true or false for the implementation.

Since POSIX standards usually are written in prose style, it usually does not suffice to copy a statement directly from the POSIX standard. Each assertion shall be a stand-alone statement. Assertions shall be worded so that **PASS** (see Section 7) is the result code for a conforming implementation.

A definitive statement is a statement in a standard that contains or implies a *shall*, *should*, or *may*. This section, unless superseded by a POSIX.n standard, describes how statements in a POSIX standard using *shall*, *should*, and *may* are interpreted to arrive at the assertions.

A statement in the POSIX standard that applies to the implementation and either includes the word *shall* or is a declarative sentence that implies the verb *shall* is interpreted as a requirement for the implementation. Such a statement by itself, or in conjunction with other definitive statements, corresponds to an assertion.

A statement that applies to a conditional feature for the implementation, that uses the verbs *should* or *may*, or is a statement that implies the verbs *should* or *may*, is interpreted as a requirement on the implementation, if the implementation supports the option. Such a statement by itself or in conjunction with other definitive statements corresponds to an assertion.

No assertions shall be written for a statement that applies only to usage of an implementation rather than to an implementation itself.

No assertion shall be written for a statement that uses the verbs *should* or *may* when that statement is either a warning to programmers or an implementation recommendation.

6.2 Assertion Constructs

6.2.1 Assertion Parts

There are three parts to an assertion: the assertion number, the assertion classification, and the actual text for the assertion.

6.2.1.1 Assertion Number

Each assertion is assigned an assertion number that is unique within the set of assertions for a particular element. The assertions should appear in the same order as the corresponding statement in the POSIX standard. They should be numbered sequentially beginning with one (1) in each element.

New or modified assertions shall be numbered sequentially beginning with the next available number. If changes to the assertion list result in unused assertion numbers, those numbers are indicated at the end of the assertion list for each element preceded by the phrase **Unused Assertion Numbers:**.

6.2.1.2 Assertion Classification

If an assertion corresponds to a statement that is based on a conditional feature, it is classified as an assertion with type **C** or **D**, and it is written as an 'If' statement. Otherwise, it is classified as an assertion with type **A** or **B**.

If an assertion corresponds to a statement that can be tested by a portable PCTS in a reasonable amount of time and effort, it is classified as a base assertion (type **A** or **C**). Otherwise, it is classified as an extended assertion (type **B** or **D**). The reason why it is an extended assertion is provided as a number that corresponds to at least one of the reasons listed in Section 5.

For most assertions, the assertion classification is simply **A, B, C,** or **D** (see the matrix in Section 5). Sometimes the classification of base or extended is dependent on whether or not a qualifier is true for the implementation under test. In these cases, the assertion classification is written as either (**QUALIFIER?A:B**) or (**QUALIFIER?C:D**). In the first case, if **QUALIFIER** is true, the required assertion is classified as **A** (base); otherwise, the required assertion is classified as **B** (extended). In the second case, if **QUALIFIER** is true, the assertion for the conditional feature is classified as **C** (base); otherwise, the conditional assertion is classified as **D** (extended).

6.2.1.3 Assertion Text and Examples

The actual text for each assertion follows one of the constructs described in this subclause.

In the simplest case, the assertion is a simple statement.

Example:

01(A) The sky is blue.

If the assertion requires the environment to be in a particular state, a 'When' clause is included, followed by the word 'then'.

Example:

02(A) When there are no clouds, then the sky is blue.

If the assertion is for a conditional feature, an 'If' clause containing the conditional POSIX behavior is included, followed by a colon (':').

Example:

03(C) If the implementation provides C Standard Language-Dependent system support:
The sky is blue.

In the case where an assertion for a conditional feature requires the environment to be in a particular state, the above two constructs are combined.

Example:

04(C) If the implementation provides C Standard Language-Dependent system support:
When there are no clouds, then the sky is blue.

If an assertion is for a required feature that has different behavior depending on whether or not a particular condition is true on the implementation, an 'If ... Otherwise ...' construct is used, and its classification is either **A** or **B.**

Example:

05(A) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
The sky is blue.

Otherwise:
The sky is gray.

A qualifier is used when the classification of the assertion depends upon a feature of the implementation. The following is an example of a conditional assertion that has a qualifier to determine whether or not it is base or extended:

Example:

06({_POSIX_JOB_CONTROL}?C:D) If the implementation provides C Standard Language-Dependent system support:
The sky is blue.

If a feature or behavior is similar across multiple functions defined in the same section, the 'For' construct is allowed as a method of maintaining consistent assertion numbers among functions.

Example:

07(C) For earth():
If the implementation provides C Standard Language-Dependent system support:
When there are no clouds, then the sky is blue.

For moon():
If the implementation provides C Standard Language-Dependent system support:
When there are no clouds, then the sky is black.

When an aspect of an implementation depends on two or more conditional features, the assertions are written as multiple assertions that cover the applicable combinations of these conditional features.

Example:

08(C) If the general terminal interface and the behavior associated with {_POSIX_JOB_CONTROL} are supported:
The sky is blue.

09(C) If the general terminal interface and the behavior associated with {_POSIX_JOB_CONTROL} are not supported:
The sky is gray.

10(C) If the general terminal interface is supported and the behavior associated with {_POSIX_JOB_CONTROL} is not supported:
The sky is black.

If an assertion is based on a value that is too large to test or is indeterminate according to a POSIX standard, a symbol beginning with the characters "PCTS" and representing the testing limit for that value is used in the assertion. The assertion describes the expected behavior of the feature based on the value of the "PCTS_" symbol. This symbol is used by a PCTS to determine the expected results for the assertion.

Example:

11(A) If {OPEN_MAX} is defined and {OPEN_MAX} < {PCTS_OPEN_MAX}:
The sky is blue.

Otherwise:
The sky is gray.

When an assertion requires many tests in order to test the assertion thoroughly, a list or table is provided that contains the minimum required tests.

Example:

12(A) The sky has the colors black, white, aqua blue, and slate blue.

Or:

12(A) The sky changes colors during the day as shown in Table 6-1.

Table 6-1 – Changing Sky

Time of Day	Color
Dawn	black
Midday	white
Sunset	aqua blue
Evening	slate blue

Two assertions are written when a statement in a POSIX standard specifies a set of independent alternative setup conditions, some of which cannot be set. A base assertion is written for the conditions that can be set. An extended assertion is written for the conditions that cannot be set.

Example:

13(B) When it is noon, or the earth is struck by an asteriod, the system prints "Hello there."

Splits into the following two assertions:

13(A) When it is noon, the system prints "Hello there."

14(B) When the earth is struck by an asteroid, the system prints "Hello there."

6.2.1.3.1 General Assertions

General assertions are statements of behavior or functionality derived from a POSIX standard that apply to more than one element and expand into one assertion specific to each applicable element. General assertions are written as GA#, where # is an integer beginning with one (1) and the value of # is unique within each POSIX standard. General assertions are not assigned a classification. Each assertion that is derived from a general assertion contains a reference to the general assertion from which it was derived.

Example:

GA1 When a special key is pressed, the system stops.

This assertion becomes the following for the BREAK key element:

15(A) When the BREAK key is pressed, the system stops. (See GA1)

6.2.1.3.2 Reference Assertions

Reference assertions are written as **R#**, where # is an integer beginning with one (1) and the value of # is unique within the description of each element. Reference assertions are not assigned a classification. Reference assertions contain pointers to the actual assertion(s).

Example:

R01 When there are no clouds overhead, then the sky is blue. (See Assertion(s) 1,2 in 6.2.1.3)

6.2.1.3.3 Testing Requirements

When there is a possibility that an assertion may be ambiguous or subject to misinterpretation, a clarification of the assertion is provided by adding a testing requirement statement after the assertion. This is necessary when attempts to provide additional precision to an assertion only results in increasing its ambiguity and "fog factor."

Example:

16(A) When clouds cover the sky, the sky is gray.

Testing Requirement(s):
Test for initial sky colors of black, white, aqua blue, and slate blue.

6.2.1.3.4 Unused Assertion Numbers

"Unused Assertion Numbers: # —" denotes that assertion number # is no longer valid because the wording has been amended in the standard under test or the original assertion was determined to be incorrect.

6.2.2 Assertion Format

This subclause formally defines the content of assertions using a formal specification. This formal specification is used to assure a clarity and preciseness in the definition of the assertion constructs that will facilitate a consistent definition of assertions by future test methods working groups.

The following rules were used in developing the formal specification of the assertion format:

(1) A variable name is a single word consisting of uppercase letters and digits.

(2) Variables are defined by a variable name followed by the symbol ::= and the definition.

(3) A definition consists of ordered sets of variables, literals, and variable text.

(4) Quoted (" ") strings are literals that shall be used in assertions exactly as specified.

(5) A variable may have more than one definition. The symbol || is used to delineate alternatives.

(6) Variable text is specified by an unquoted phrase in uppercase and lowercase.

All assertions shall be written in a form consistent with the grammar and rules given in 6.2.2.1 and 6.2.2.2.

6.2.2.1 Assertion Format Grammar

ASSERTION::=	Number "(" CLASS ")" STATEMENT || GENASSERT || REFASSERT
CLASS::=	CLASSLETTER || ICTEXT "?" CLASSPAIR
CLASSLETTER::=	"A" || "B" || "C" || "D"
CLASSPAIR::=	"A:B" || "C:D" || CLASSLETTER":UNUSED"
STATEMENT::=	MAINSTATEMENT || FORSTATEMENT
MAINSTATEMENT::=	ATEXT || A1 ATEXT || A1 ATEXT "Otherwise" A4 ATEXT || A2 ATEXT
A1::=	A3 || A3 "when" SCTEXT NEXTSTATE "then"
A2::=	"when" SCTEXT NEXTSTATE "then"
A3::=	"If" ICTEXT NEXTCONDITION ":"
A4::=	NULL || STATE "then"
STATE::=	"when" SCTEXT NEXTSTATE
NEXTSTATE::=	NULL || "and" SCTEXT NEXTSTATE || "or" SCTEXT NEXTSTATE
NEXTCONDITION::=	NULL || "and" ICTEXT NEXTCONDITION
ATEXT::=	Assertion text
ICTEXT::=	Implementation Condition text
SCTEXT::=	State Condition text
GENASSERT::=	"GA" Number STATEMENT
REFASSERT::=	"R" Number STATEMENT
FORSTATEMENT::=	"For" ELEMENTLIST ":" MAINSTATEMENT || FORSTATEMENT blankline "For" ELEMENTLIST ":" MAINSTATEMENT
ELEMENTLIST::=	Element name || Element name "," ELEMENTLIST

6.2.2.2 Assertion Format Rules

(1) For type A and B assertions, an 'If' must always be followed by an 'Otherwise'.

(2) Type C and D assertions must start with an 'If'.

(3) For type C and D assertions, an 'Otherwise' shall not be used.

(4) Assertions derived from a general assertion (GENASSERT) must include a reference to the general assertion in the form (see GA# in x.y.z) following the MAINSTATEMENT (see 6.2.1.3.1).

(5) A reference assertion (REFASSERT) must include a pointer in the form (see Assertion(s) # in x.y.z) following the MAINSTATEMENT (see 6.2.1.3.2).

(6) Testing requirements shall follow the MAINSTATEMENT and be preceded by a blank line (see 6.2.1.3.3).

Section 7: Assertion Test Output

7.1 Test Result Codes

After execution of assertion tests, there shall be sufficient output to derive the following information:

- The element name tested,
- The number of the assertion tested,
- Whether the corresponding assertion is base or extended, and
- The test result code.

There are two types of test result codes: final and intermediate.

The final test result codes are:

- **PASS** - The assertion was found to be true based on the execution of its assertion test.
- **FAIL** - The assertion was found to be false based on the execution of its assertion test.
- **UNTESTED** - Either there is no assertion test or the implemented test(s) for the extended assertion is not complete enough to result in a test result code of **PASS** or **FAIL**.
- **UNSUPPORTED** - An assertion test could not be performed because the conditional feature was not implemented. Additional final test result codes shall only be used in situations for which none of the above final test result codes apply. Any additional final test result codes used by a test method shall be documented in the test method documentation.

The intermediate test result code is:

- **UNRESOLVED** - At least one of the following conditions is true:
  (1) The assertion test requires manual inspection in order to determine its result.
  (2) The setup for the assertion test did not complete in the manner expected.
  (3) The test program containing the assertion test was unexpectedly interrupted.
  (4) The assertion test could not be executed because a previous assertion test on which it depended failed.
  (5) The test program containing the assertion was not initiated.

(6) Compilation or execution of the test program produced unexpected errors or warnings.

(7) The assertion test did not resolve to a final test result code for any other reason.

All occurrences of **UNRESOLVED** test result codes shall resolve to one of the final test result codes before a statement of conformance (see Section 8) is made.

The permissible test result codes for each class of assertions as explained in Classification of Assertions (see Section 5) are shown in Table 7-1.

Table 7-1 – Test Result Codes

	Base Assertion	Extended Assertion
Required Feature	PASS	PASS
	FAIL	FAIL
	UNRESOLVED	UNTESTED
		UNRESOLVED
Conditional Feature	PASS	PASS
	FAIL	FAIL
	UNSUPPORTED	UNSUPPORTED
	UNRESOLVED	UNTESTED
		UNRESOLVED

Test methods shall not attach any other meaning to the test result codes described above.

For each element, there is a largest assertion number that is actually used. For each number less than that maximum and for which there is no valid assertion, a test method shall output **UNUSED** in place of a valid test result code.

When the test result code is **FAIL** or **UNRESOLVED**, the test method shall provide, to the extent possible, sufficient information to determine the cause of the result.

If an unexpected event occurs while determining the result of an assertion test that could invalidate the test result, the test method shall identify it to the extent possible and report such an occurrence. If the assertion test cannot be completed after such an occurrence, the test result code shall be **UNRESOLVED**. If the test method is a PCTS, it should strive to continue to execute test programs, but shall report any assertion test that could not be executed as **UNRESOLVED**. A PCTS should attempt to recover from the unexpected event in a manner that maximizes the execution of assertion tests while ensuring the validity and correctness of subsequent test results.

Section 8: Statement of Conformance to a POSIX Standard

8.1 Conformance Statement Contents

The results of the execution of the test methods against a specific target system may be summarized in a statement of conformance concerning that target system. If a statement of conformance is made, the following information shall be provided in the conformance statement:

- The name and edition of the POSIX.n standard to which conformance is being measured;
- The names and version numbers of the test methods used;
- The POSIX.n test method specifications to which the test methods conforms;
- The name, model, and configuration of the computer systems tested and the name, version, and release level of the implementation stated in terms of the identification scheme of the implementor;
- The name and version of the audited PCD (if a PCD is required by the POSIX.n standard);
- The date the test methods were applied; and
- The language bindings that have been tested.

In addition, the following information shall be available:

- The final result code for each assertion test, and
- A description of any modifications made to the test methods.

A statement of conformance to a POSIX standard is based on a completed test of the target system using a set of POSIX.3 conforming test methods, where for each POSIX.3 assertion for that standard, there is a correctly assigned test result code. A target system conforms to that standard if those test result codes are as in Table 8-1.

Table 8-1 – Conformance Test Result Codes

	Base Assertion	Extended Assertion
Required Feature	PASS	PASS UNTESTED
Conditional Feature	PASS UNSUPPORTED	PASS UNSUPPORTED UNTESTED

Annex A
(informative)

Bibliography

This Annex contains lists of related open systems standards and suggested reading on historical implementations and application programming.

A.1 Related Open Systems Standards

A.1.1 Networking Standards

{B1} ISO 7498: 1984, *Information processing systems—Open Systems Interconnection—Basic Reference Model.*[1)]

{B2} ISO 8072: 1986, *Information processing systems—Open Systems Interconnection—Transport service definition.*

{B3} ISO/IEC 8073: 1988, *Information processing systems—Open Systems Interconnection—Connection oriented transport protocol specification.*[2)]

{B4} ISO 8326: 1987, *Information processing systems—Open Systems Interconnection—Basic connection oriented session service definition.*

{B5} ISO 8327: 1987, *Information processing systems—Open Systems Interconnection—Basic connection oriented session protocol definition.*

{B6} ISO 8348: 1987, *Information processing systems—Data communications—Network service definition.*

{B7} ISO 8473: 1988, *Information processing systems—Data communications—Protocol for providing the connectionless-mode network service.*

{B8} ISO 8571: 1988, *Information processing systems—Open Systems Interconnection—File Transfer, Access and Management.*

{B9} ISO 8649: 1988, *Information processing systems—Open Systems Interconnection—Service definition for the Association Control Service Element.*

1) ISO documents can be obtained from the ISO office, 1, rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse.

2) IEC documents can be obtained from the IEC office, 3, rue de Varembé, Case Postale 131, CH-1211, Genève 20, Switzerland/Suisse.

{B10} ISO 8650: 1988, *Information processing systems—Open Systems Interconnection—Protocol specification for the Association Control Service Element.*

{B11} ISO 8802-2: 1989 [IEEE Std 802.2-1989 (ANSI)], *Information processing systems—Local area networks—Part 2: Logical link control.*

{B12} ISO 8802-3: 1989 [IEEE Std 802.3-1988 (ANSI)], *Information processing systems—Local area networks—Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications.*

{B13} ISO/IEC 8802-4: 1990 [IEEE Std 802.4-1990 (ANSI)], *Information technology—Local area networks—Part 4: Token-passing bus access method and physical layer specifications.*

{B14} ISO 8802-5: ... (IEEE 802.5-1989), *Information technology—Local area networks—Part 5: Token ring access method and physical layer specifications.*

{B15} ISO 8822: 1988, *Information processing systems—Open Systems Interconnection—Connection oriented presentation service definition.*

{B16} ISO 8823: 1988, *Information processing systems—Open Systems Interconnection—Connection oriented presentation protocol specification.*

{B17} ISO 8831: 1989, *Information processing systems—Open Systems Interconnection—Job transfer and manipulation concepts and services.*

{B18} ISO 8832: 1989, *Information processing systems—Open Systems Interconnection—Specification of the basic class protocol for job transfer and manipulation.*

{B19} CCITT Recommendation X.25, *Interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCT) for terminals operating in the packet mode and connected to public data networks by dedicated circuit.*[3)]

{B20} CCITT Recommendation X.212, *Information processing systems—Data communication—Data link service definition for Open Systems Interconnection.*

A.1.2 Language Standards

{B21} ISO 1539: 1980, *Programming languages—FORTRAN.*

{B22} ISO 1989: 1985, *Programming Languages—COBOL.*

{B23} ISO 8652: 1987, *Programming Languages—Ada.*

3) CCITT documents can be obtained from the CCITT General Secretariat, International Telecommunications Union, Sales Section, Place des Nations, CH-1211, Genève 20, Switzerland/Suisse.

{B24} ANSI X3.113-1987[4], *Information systems—Programming language—FULL BASIC.*

{B25} ANSI/IEEE 770X3.97-1983, *Standard Pascal Computer Programming Language.*

{B26} ANSI/MDC X11.1-1984, *Programming Language MUMPS.*

A.1.3 Graphics Standards

{B27} ISO 7942: 1985, *Information processing systems—Computer graphics—Graphical Kernel System (GKS) functional description.*

{B28} ISO 8632: 1987, *Information processing systems—Computer graphics—Metafile for the storage and transfer of picture description information.*

{B29} ISO/IEC 9592: 1989 (ANSI X3.144-1988), *Information processing systems—Computer graphics—Programmer's hierarchical interactive graphics system (PHIGS).*

A.1.4 Database Standards

{B30} ISO 8907: 1987, *Database Language—NDL.*

{B31} ISO 9075: 1987, *Database Language—SQL.*

A.2 Other Standards

{B32} ISO 639: 1988, *Code for the representation of names of languages.*

{B33} ISO 3166: 1988, *Code for the representation of names of countries.*

{B34} ISO 8859-1: 1987, *Information Processing—8-bit single-byte coded graphic character sets—Part 1: Latin alphabet No. 1.*

{B35} ISO 9127: 1988, *Information processing systems—User documentation and cover information for consumer software packages.*

{B36} ISO/IEC 9945-2: . . . ,[5] *Information technology—Portable operating system interface (POSIX)—Part 2: Shell and utilities.*

{B37} ISO/IEC 10646: . . . ,[6] *Information processing—Multiple octet coded character set.*

{B38} IEEE Std 100-1988, *IEEE Standard Dictionary of Electrical and Electronics Terms.*

---

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

5) To be approved and published.

6) To be approved and published.

Annex B
(informative)

Rationale and Notes

B.1 General

The developers of this International Standard rejected any requirements on how to create a test suite. It was recognized that target systems may range from small embedded systems to a large mainframe. It was also recognized that some company might have a specific situation where they would like to conform to this International Standard, but the situation required a nonportable test method.

Automatic testing versus interactive testing was discussed. It was concluded that the test methods could include interactive testing, although the goal should be automatic testing in a thorough manner. This conclusion was reached even though it was recognized that this could put a requirement on having additional hardware, such as having two asynchronous ports versus one, so that closed-loop testing could be performed.

The meaning of a PCTS and whether it included a documentation audit of the required documentation was discussed. The developers of this International Standard concluded that the documentation audit was separate from the PCTS.

B.2 Terminology and General Requirements

The terms defined in this section are unique to this International Standard and apply to other POSIX.3.n standards.

assertion

The developers of this International Standard earlier defined an assertion to be the essential functionality from a statement in an applicable POSIX standard that would correspond to a single test result. This would permit a PCTS developer to execute a specific software or procedure test and documentation audit on a conforming implementation to verify the functionality of that assertion. Use of the phrase "software or procedure" is used in favor of "software" to cover assertions that require a manual procedure to determine results.

The developers of this International Standard later decided to drop the concept of correspondence of an assertion to a single test result from the definition. This was primarily because the previous definition imposed an unnecessary restriction on the implementation of a PCTS. Moreover, the definition disagreed with one in IEEE Std 729-1983, which is one of the referenced documents of this International Standard.

assertion test

This is defined to be the software or procedure that determines conformance of a POSIX implementation to an assertion. This would include, for example, an executable software test, the manual verification inspection of output, or the auditing of required implementation-defined documentation required by a POSIX standard.

base assertion

This is an assertion that is required to be tested because a POSIX standard required it as a feature with a shall statement or through an implied shall statement in a declarative sentence. Another class of base assertion is where a conditional feature (see 2.2.2.5) has been implemented and therefore must be tested.

conditional feature

In POSIX.3, the term *conditional,* which has a broader meaning, is used in favor of the POSIX.1 term *optional.* The term optional may be defined in other POSIX standards, but their definitions will always be subsets of the POSIX.3 term conditional. The term optional, in a POSIX standard, implies a feature or behavior that the developers of the applicable POSIX standard have deliberately specified as an option, as opposed to a requirement. The issue is raised when a statement in the standard states that a feature is only required if it is actually implemented by the conforming POSIX system, which means that within a PCTS a condition is raised as to whether the test should proceed if the relative POSIX feature has been implemented. Examples of this are asynchronous communications, file types, and privileged mode in POSIX.1.

development system

It is not necessary for the development system to be POSIX conforming. However, it shall have all the software necessary to generate the configured PCTS for the specified target system. In such a case, the compilation environment used to generate the configured PCTS will be considered part of the target system configuration.

extended assertion

This is an assertion that is not a base assertion, hence it is not required to be tested. The reason the developers of this International Standard created this class of assertion is because there will be features specified in the POSIX standards that will be untestable (see B.5).

PCTP

This standard defines PCTS to be software. To test an assertion requires executing the assertion test software. However, the result produced by the assertion test software may not resolve to the final result code of the assertion test. A procedure may have to be manually executed to resolve the assertion test to a final **PASS** or **FAIL** test result code. In other words, it takes both a piece of software and human executed procedures to produce a final result for some assertion tests.

For example, an element such as the `lp` command may have an assertion defined that requires the printing of some specific characters on a printer. The assertion test software can send the characters to the printer, but it cannot verify that the printer printed the correct characters. In this

example, the test result code of the assertion test software would be an intermediate test result code such as **UNRESOLVED.** The test result message would indicate that someone must follow a procedure to verify that the printer printed the correct characters. In this example, both software and manual procedures are required to resolve the assertion test to a test result code of **PASS** or **FAIL.**

In conclusion, the final results needed to produce the conformance statement might require software testing and human executed procedures.

required feature

This is a feature that is required by a conforming implementation as specified by the applicable POSIX standard. A required feature is determined by a specific shall statement or an implied shall statement in the POSIX standard. A required feature usually generates a base assertion, but it may become an extended assertion if the feature is determined to be untestable by a PCTS with supporting rationale as discussed in B.5.

This definition of required feature handles the 'If ... Otherwise ...' assertion construct. The alternative to this would have been to require that 'If ... Otherwise ...' assertions would have to be split up into two "C" type assertions, where one of the two would result in an **UNSUPPORTED** test result code. In order to avoid this and to preserve the concept that in the absence of a conditional feature there is required behavior for a conforming implementation, the definition of required feature was modified.

Some terms that were in previous drafts of this International Standard have been removed. However, the rationale for these definitions has been preserved for historical reasons.

closed-loop mode

A closed-loop mode of terminal I/O testing of a target system configuration would be where two target system asynchronous ports are electrically connected in such a way that data communications is performed between the ports using modem control protocols.

compliance

The term *compliance* was introduced in POSIX.3 to provide an efficient way to represent specific, acceptable levels of conformance of an implementation to a POSIX standard (as measured by a POSIX.3 conforming test method). Thus, "compliant with" a POSIX standard as measured by a test method means "passing" that test method.

The standard dictionary meanings of *compliance* and *conformance* are very similar. The developers of this International Standard chose to differentiate the meanings of these words in the context of measuring conformance to a POSIX standard in order to distinguish between the theoretical ideal of complete conformance and the practical measurement of that conformance.

Based on the definitions of conformance in a specific POSIX standard, a supplier may claim conformance of an implementation to that POSIX standard. Similarly, based on the definition of conformance in the POSIX.3 standard, a test methods supplier may claim conformance to the POSIX.3 standard. The purpose of the test method specified in the POSIX.3 standard is to measure

the level of conformance of an implementation to a POSIX standard. Practical limitations, however, preclude the test method from guaranteeing complete conformance. It follows that an implementation might be 100% compliant with a POSIX standard, even though it is less than 100% conformant to that standard.

To sum up, an implementation may conform to a POSIX standard to some degree, but there is no way of guaranteeing complete conformance, and no efficient way of describing acceptable degrees of conformance without introducing additional terms.

conformance

After the draft balloting of this International Standard, the developers of this International Standard decided that the distinction between the terms *compliance* and *conformance* should be eliminated as it was causing confusion. The developers of this International Standard decided to treat both terms synonymously as done by English dictionaries, and to use the term *conformance* to ascertain the application portability of a computing environment.

B.3 Test Methods Conformance to This Standard

The developers of this International Standard determined that it was necessary to require the developers of a PCTS to document the nature of their test suite. The developers of this International Standard did not specify the exact nature of the required PCTS documentation, as the marketplace would determine the needs of the user for understanding each PCTS to test conforming systems. The developers of this International Standard did provide the requirement that a PCTS shall include all the information to install, configure, and execute the PCTS. The other requirement was that a user of the PCTS should be able to gather and interpret the results of the PCTS. The developers of this International Standard also specified that the documentation would meet the requirements of a Conforming POSIX Application as specified in POSIX.1.

While the standard states that the documentation should provide information as to how to execute the PCTS, it is implicit that any PCTS may only be able to execute on a subset of the totality of conforming implementations. PCTS implementations are likely to have dependencies on facilities outside the POSIX standard under test, and the documentation only needs to describe installation and execution methods on an environment for which the PCTS was designed.

As an example for items to consider, the developers of this International Standard provided what they felt would be a sample of information that a PCTS documentation package could supply with the PCTS product.

The documentation for the test methods may contain the following information:

— The edition of the standard being tested;

— An overview of the PCTS, including the extended assertions and the optional POSIX features tested by the PCTS;

— A specification of any deviance from the recommendations defined within this International Standard;

- A specification of any known limitations of the PCTS;
- A specification of any release-specific change;
- A specification of any configuration-specific parameters;
- A specification of development system hardware requirements such as memory, disk space, terminal, and printer needs;
- A specification of development system software requirements;
- A specification of the hardware and software requirements necessary to execute the PCTS;
- A definition of the user interface syntax;
- A description of the procedures for transferring the PCTS to a target system, if the PCTS requires such a transfer;
- A specification of the format of the PCTS output;
- Examples of sample PCTS output;
- A description of the procedures for obtaining the PCTS output;
- Examples of the procedures for obtaining PCTS output;
- A list of any known assertion tests that may severely impact PCTS execution; and
- A description of any PCTS trouble-clearing procedures.

B.4 Testing Levels and Complexity Levels

B.4.1 Introduction

There is no additional rationale provided for this subclause.

B.4.2 Testing Levels

The developers of this International Standard determined it was necessary to define what the testing levels were when developing a PCTS. This was put into the actual standard to provide a guideline for those entities developing PCTSs for the POSIX standards. The degrees of testing were *exhaustive, thorough,* and *identification,* with discussion of each methodology contained in the standard.

The conclusion of the developers of this International Standard was that each PCTS would vary and each level of testing for each assertion would also vary.

B.4.3 Complexity Levels

A software problem exists within the context of what exactly are the ramifications of an element that should be tested. What are the practical and economic considerations? The components of an element and the interrelationships of that element to other elements are what can create the complexity, from a design point of view, within a PCTS.

Each element has at least two realms of complexity. One realm deals with the complexity of the functionality of the element itself. The other realm is how hard it is to test the functionality of the element. An element may have much functionality and many hundreds of assertions must be defined to test all aspects of the element, but the testing of the element is rather simple, while another element may be simple in functionality, but the testing is not simple.

The developers of this International Standard chose to only deal with the one realm. This is the realm that deals with the complexity of the functionality of the element.

B.4.4 Conclusion

The developers of this International Standard considered, and rejected, a proposal to remove this clause from the standard. The objection raised was, "this section has no measurable requirement on a PCTS and the terms are subjective." This is true. However, while there is nothing measurable in this clause, much of POSIX.3 deals with the definition of general concepts, and other standards exist that present such general concepts. For example, ISO/IEC DIS 9646-1: ..., *Information technology—OSI conformance testing methodology and framework—Part 1: General concepts,*[1)] discusses general conformance testing concepts as they apply to OSI interfaces. Because of these existing documents, this objection was rejected.

This section specifically deals with the concept of conformance testing and the evaluations of elements that must be made by the assertion writers and PCTS writers.

B.5 Classification of Assertions

It is highly recommended that extended assertions be tested whenever possible. Development of assertion tests for extended assertions is encouraged because it is desired that test suites be made available that are as complete as possible in regards to the assertions in POSIX.3.n.

Seven reasons exist for extended assertions.

(1) There is no known portable test method for this assertion.

The first reason states that a portable test cannot be written for all conforming systems because the actual software may change for different

1) *To be approved and published.*

target systems that the PCTS supports. This is the case where exits into the hosting operating system would have to be written for a POSIX conforming system and would most likely be different for each target system. Reason 1 is used instead of reason 3 when there is a known test method for the assertion for at least one POSIX implementation, but that method is not portable to all POSIX implementations.

(2) The corresponding statement in the POSIX standard to which conformance is being measured is not specific enough to write a portable test.

The second reason stated is that the POSIX standard was not clear in the specification of a feature from a software testing viewpoint. An example of this is in POSIX.1, where it is stated that the *stat.h* data items shall have meaningful values. A PCTS would have great difficulty in determining the test software to verify that a *stat* structure contains meaningful values. It may be that supplements developed for the POSIX standards can address these statements in the applicable POSIX standard.

(3) There is no known reliable test method for this assertion.

The third reason stated is the result of the developers of this International Standard being unable to determine a software or procedure test to determine if the assertion did what was stated in the applicable POSIX standard. An example of this case can be seen when trying to determine elapsed clock time for a particular function in a POSIX standard.

(4) Testing the assertion requires setup procedures that involve an unreasonable amount of effort by the user of a PCTS.

(5) Testing the assertion would require an unreasonable amount of time or resources on most systems.

(6) Creating an assertion test would require an unreasonable amount of test development time.

The fourth, fifth, and sixth reasons were a judgment call on the part of the developers of this International Standard trying to perceive the system, engineering, and business costs of testing assertions on a conforming POSIX system. The developers of this International Standard achieved consensus on the aspects of what the word "reasonable" denotes in POSIX.3 regarding testing assertions. The working group also realized that the marketplace would drive the need for extended assertions to be tested if they were critical within the respective industry.

(7) The assertion test could have an adverse effect on completion of a test method.

The classification of assertions, the assertion matrix, and the extended assertion rationale can be used to categorize an assertion in the POSIX.n standard.

This standard does not preclude a test suite from containing further tests beyond those contained in the assertion list for the corresponding POSIX standard. Where a test suite contains such assertions, the same test result codes may be used as defined in Section 7, but the additional assertions must not be taken into account in determining the conformance of an implementation to the POSIX standard.

B.6 Writing Assertions

B.6.1 Assertion Determination

Certain statements containing the word *may* do not provide sufficient clarity. For these cases no assertion can be written. For example, the statement

> A call to *fopen*() may cause the *st_atime* field of the underlying file to be marked for update.

does not declare whether the behavior is to be consistent for all file types or a specific implementation.

The definition for may is the most difficult to deal with when writing a test suite. Each may could indicate a branch in the flow of control of a function, the definition of which is at the discretion of the supplier. From the perspective of a test suite, a reliable test that exercises a may feature is difficult to create. The test might attempt to use the feature and report the results, but it cannot report an error if the feature acts in a nondeterministic way (from the point of view of the standard). Unfortunately, an allowable side effect of a may feature could be catastrophic.

B.6.2 Assertion Constructs

Very late in the development of this International Standard, it was recognized that constructs must be provided for those times when a standard states that the concept in question is implementation defined and nothing in the standard requires that implementation to work, yet the concept is required to be implemented in order to test that assertion. A case in point is read-only file systems. The POSIX.1 standard does not tell how to create a read-only file system. The POSIX.1 standard does not require them, but allows for them. How is a file created on a read-only file system if the implementation does not support it? This, then, is an implementation condition as opposed to a condition defined by the standard.

Because of this problem, a new definition of a condition was introduced.

The developers of this International Standard now recognized that *conditions* have two realms.

— Conditions defined by a standard.

— Conditions not defined by a standard, but defined by the implementation.

Assertion constructs have been defined to handle all known conditions. We believe that these constructs can be used when writing assertions for any standard.

The POSIX standards define functionality based on qualifiers or prerequisites or, for lack of another word, conditions. If the condition is not a required feature, this International Standard (POSIX.3) uses the notation of conditional feature. If the condition is a statement where an implementation must choose between two or more qualifiers this International Standard calls this a *condition*. It is required that an implementation choose one of these conditions. These are conditions defined by the POSIX standard.

The POSIX standards allow functionality that they do not explicitly define. They state the concept as implementation defined. Some assertions may require that the implementation support the condition in order to test the assertion.

B.6.2.1 Assertion Parts

B.6.2.1.1 Assertion Number

(1) Assertion numbers are identifiers, and do not imply a testing sequence or location in the standard being tested.

(2) When an assertion is added, the next sequential unused number for the element is assigned to the new assertion. The assertion will be placed as close as possible to where it appears lexically in the standard for which the assertions are being written.

(3) When an assertion is deleted, its assertion number is put on the **UNUSED** list at the end of the assertion list for each element.

(4) If an assertion is changed substantially, it shall be deleted and a new assertion shall be added.

B.6.2.1.2 Assertion Classification

There is no additional rational for this subclause.

B.6.2.1.3 Assertion Text and Examples

This section shows examples of writing assertions. They are simplistic in scope, but they accurately reflect the way assertions are expressed in POSIX.n standards.

The 'For' construct may be used within general assertions. For example:

GA1 For all elements, except *asteroids*() and *moons*():
When a special key is pressed, the system stops.

This assertion becomes the following for the BREAK key element:

15(A) When the BREAK key is pressed, the system stops. (See GA1)

B.7 Assertion Test Output

To aid in the test result analysis process, it was decided that test output must appear for each assertion, whether or not it is tested. The output for those test assertions that are not tested because they are extended assertions for which no tests are available show an **UNTESTED** test result code. The **UNTESTED** test result code is intended to be used whenever an assertion test is not provided. If an assertion test is provided but not attempted because of some environmental failure, the **UNRESOLVED** test result code is used.

In an earlier draft, **PASS** was defined as a successful test. This was changed because in software testing, the term *successful* often means that a bug was found in the feature. This is just the opposite of what was to be portrayed in the definition of **PASS.**

For several drafts this chapter contained the **ALERT** test result code with the following definition:

— The extended assertion was found to be false on the target system based on execution of its assertion test.

Also, **FAIL** was defined for base assertions only. But the developers of this International Standard decided that **FAIL** should apply when the assertion is found to be false, regardless of whether or not it is a base or an extended assertion. Some desired the means to determine from the test output whether or not a FAILed test was from a base or an extended assertion, now that this distinction had been taken away from the test result code. So the decision was made to require an indication as to whether or not the assertion is base or extended in the test output for each assertion test.

Here is an example of how test output conforming to this International Standard might look:

```
fork   3 BASE   PASS   Child ID different from parent ID
fork   4 EXTEND FAIL   Child ID matched an active ID
```

There was considerable debate about the ability to resolve a test result code of **UNRESOLVED** to a test result code of **UNTESTED** in the case of an extended assertion that relied upon a supporting facility. It was agreed by the developers of this International Standard that a failure to supply such a support facility would cause the final result code to be represented as **UNTESTED,** since the test had not been performed (and possibly should not be performed) by the system under test.

During this debate the question of reliance of assertions on implementation-defined features was also considered. If a base assertion is incorrectly classified, it can be reclassified following the standard IEEE process. It was suggested that such features could be provided by an implementation in such a manner as to prohibit a test from accessing these features. (The particular example cited was that of obtaining a controlling terminal.) A review of the definition of "implementation defined" in POSIX.1 led to the belief that the conformance document needed to detail the "correct program construct and correct data" that was required when using such a feature. This supported the view that all implementation-defined features needed to be available to a test author in an accessible manner on all conforming implementations.

The paragraph on unexpected events is included in the standard because the developers of this International Standard felt that a standard PCTS should have the ability to recover from unexpected events such as signals. While it is true that there are limitations to how well a piece of software can recover from such events, the PCTS software is expected to recognize and report these events whenever possible.

The conformance testing reference model can be represented by two inverted trees (see Figure B-1). One tree is the PCTS and the other tree is the procedures.

The forks in the tree are:

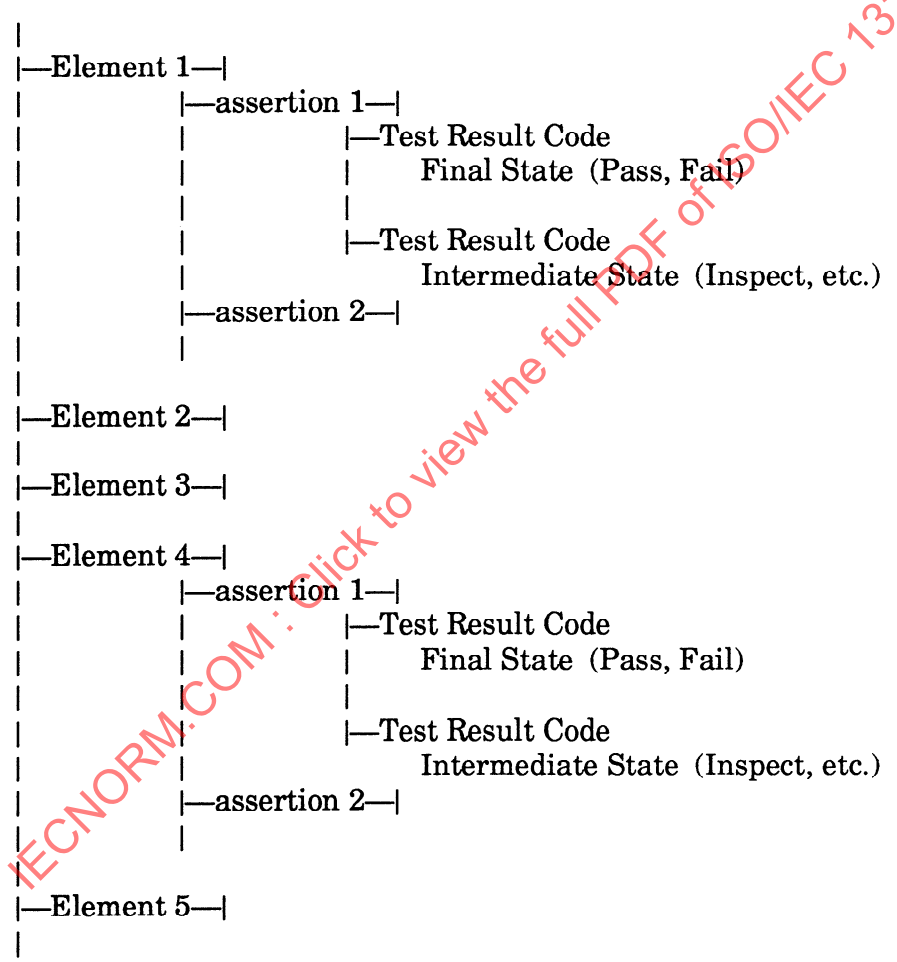
— Element being tested

— Assertions being tested

— Test result codes (Final or Intermediate)

The lowest limb of the tree in most cases will result in an assertion test conclusion, and the test result code will be final. The test result code can be used as evidence in the preparation of a conformance statement. However, there are cases where the lowest limb cannot resolve to a final state. In this case, the lowest limb will represent an intermediate test result code. To determine the final test result one must return to the top of the element tree until one can resolve to a final state.

Figure B-1 – Software Testing and Procedure Testing



B.8 Statement of Conformance to a POSIX Standard

There was a concept of "Extended Compliance" in earlier drafts of this International Standard. It relied on calling **FAIL**ures of Extended Assertions an **ALERT** test result code. Here is its previous definition:

Extended Compliant: A specific target system is extended compliant if it is base compliant [where base compliant is the current definition of compliant] and the following conditions are true:

— At least one extended assertion of a required feature or a conditional feature resolved to a **PASS** test result code.

— For all extended assertions of required features that do not resolve to **PASS,** The test result codes resolve to either **ALERT** or **UNTESTED.**

— The test result codes for extended assertions of conditional features that do not resolve to **PASS,** resolve to **ALERT, UNTESTED,** or **UNSUPPORTED.**

The extended compliance concept was removed because it allowed a level, "extended," that could be achieved by passing only one extended assertion. The counter-case presented a challenge to credulity; The ability to have base compliance with FAILed extended assertions is not within the spirit of POSIX.1.

The issue of "buggy" test methods was properly factored out of consideration. Portability of test methods and specificity of the underlying standard (POSIX.1) are the real concerns.

The discussion considered the POSIX implementor who would "shop around" for a PCTS vendor who would have a suite that would not fail the implementation.

The following is a sample compliance statement as shown in an earlier draft of this International Standard.

> The XYZSYS Release 2.0 target system with conditional features, supplementary groups, and job control supported, is COMPLIANT with ISO/IEC 9945-1: 1990, C Language Binding (C Standard Language-Dependent System Support) as measured by the XSVS Release 1.0 PCTS, which conforms to POSIX.3 and POSIX.n.
>
> The test methods were applied from Jan 2 to Jan 10, 1990.
>
> Complete results are in "XSVS Release 1.0 Output for XYZSYS Release 2.0, Report 060189-23," XYZ Records Office, Vancouver, B.C., Canada.
>
> A list of extended assertions that were tested is provided in the attached report.
>
> No changes were made to the target system or to the PCTS during execution of the PCTS.

After initial balloting, the terms *conformance* and *compliance* were made synonymous and the working group decided to use only the term *conformance* (see B.2). Accordingly, the term *compliance* was consistently changed to *conformance.*

In addition, it was decided that the statement of conformance should include identification of the configuration version and release level of the computer