Recognized as an
American National Standard (ANSI)

IEEE
Std 2003.1-1992
(Formerly known as IEEE P1003.3.1)

IEEE Standard for Information Technology—Test Methods for Measuring Conformance to POSIX—Part 1: System Interfaces

Sponsor

Technical Committee on Operating Systems and Application Environments
of the
IEEE Computer Society

Approved December 3, 1992

IEEE Standards Board

Approved May 12, 1993

American National Standards Institute

Abstract: IEEE Std 2003.1-1992 provides a definition of the requirements placed upon providers of POSIX test methods for POSIX.1 (IEEE Std 1003.1-1990; ISO/IEC 9945-1: 1990). These requirements consist of a POSIX.1-ordered list of assertions defining those aspects of POSIX.1 that are to be tested and the associated test methods that are to be used in performing those tests. This standard is aimed primarily at POSIX.1 test suite providers and POSIX.1 implementors. This standard specifies those aspects of POSIX.1 that shall be verified by conformance test methods.

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

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

Copyright © 1993 by the
Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1993
Printed in the United States of America

ISBN 1-55937-275-3

No part of this publication may be reproduced in any form,
in an electronic retrieval system or otherwise,
without the prior written permission of the publisher.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

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 are adopted by the Institute of Electrical and Electronics Engineers without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standards documents.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992



Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

Contents

PAGE

Introduction v
Organization of This Standard v
Related Standards Activities v

Section 1: General 1
1.1 Scope 1
1.2 Normative References 2
1.3 Conformance 3
1.4 Test Methods 5

Section 2: Definitions and General Requirements 21
2.1 Conventions 21
2.2 Definitions 23
2.3 General Concepts 25
2.4 Error Numbers 30
2.5 Primitive System Data Types 31
2.6 Environment Description 31
2.7 C Language Definitions 32
2.8 Numerical Limits 34
2.9 Symbolic Constants 37

Section 3: Process Primitives 41
3.1 Process Creation and Execution 41
3.2 Process Termination 56
3.3 Signals 67
3.4 Timer Operations 93

Section 4: Process Environment 97
4.1 Process Identification 97
4.2 User Identification 98
4.3 Process Groups 106
4.4 System Identification 111
4.5 Time 112
4.6 Environment Variables 115
4.7 Terminal Identification 116
4.8 Configurable System Variables 119

Section 5: Files and Directories 123
5.1 Directories 123
5.2 Working Directory 132
5.3 General File Creation 136
5.4 Special File Creation 157
5.5 File Removal 165
5.6 File Characteristics 181

AVA Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

5.7 Configurable Pathname Variables 207

Section 6: Input and Output Primitives 219
6.1 Pipes 219
6.2 File Descriptor Manipulation 220
6.3 File Descriptor Deassignment 223
6.4 Input and Output 224
6.5 Control Operations on Files 235

Section 7: Device- and Class-Specific Functions 245
7.1 General Terminal Interface 245
7.2 General Terminal Interface Control Functions 268

Section 8: Language-Specific Services for the C Programming Language . 287
8.1 Referenced C Language Routines 287
8.2 C Language Input/Output Functions 382
8.3 Other C Language Functions 385

Section 9: System Databases 389
9.1 System Databases 389
9.2 Database Access 389

Section 10: Data Interchange Format 395
10.1 Archive/Interchange File Format 395

Annex A (informative) Bibliography 407
A.1 Related Open Systems Standards 407

Annex B (informative) Rationale and Notes 409
B.1 General 409
B.2 Definitions and General Requirements 417

Identifier Index 423

Alphabetic Topical Index 429

TABLES

Table 1-1 – Testing Constraints 10
Table 1-2 – Configuration Variables 11
Table 1-3 – Configuration Control Sequences 11
Table 1-4 – PCD Determinable Conditional Features 19
Table 2-1 – Typographical Conventions 22
Table 2-2 – C Language Limits 34
Table 2-3 – Minimum Values 35
Table 3-1 – Structure *sigaction* Members 85
Table 5-1 – *stat* Structure 181
Table 5-2 – File Owner Class Permission Masks 183
Table 5-3 – File Group Class Permission Masks 183

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

Table 5-4 – File Other Class Permission Masks 183
Table 5-5 – Access via File Owner Class 191
Table 5-6 – Access via File Group Class 191
Table 5-7 – Access via File Other Class 192
Table 6-1 – *flock* Structure 238
Table 8-1 – Standard File Descriptors 383
Table 9-1 – *group* Structure 390
Table 9-2 – *passwd* Structure 392
Table 10-1 – *tar* Format 397
Table 10-2 – *tar* Symbolic Constants 397
Table 10-3 – *tar typeflag* Field 398
Table 10-4 – *tar mode* Field 398
Table 10-5 – *cpio* Data 402

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

Introduction

(This introduction is not a normative part of IEEE Std 2003.1-1992, IEEE Standard for Information Technology— Test Methods for Measuring Conformance to POSIX— Part 1: System Interfaces, but is included for information only.)

This standard provides a definition of the requirements placed upon providers of a POSIX test method for the POSIX.1 {3} standard. These requirements consist of a POSIX.1 {3} ordered list of assertions defining those aspects of POSIX.1 {3} that are to be tested and the associated test methods that are to be used in performing those tests. This document is aimed primarily at POSIX.1 {3} test suite providers and POSIX.1 {3} implementors. This document specifies those aspects of the standard that shall be verified by conformance test methods.

Organization of This Standard

This document is organized into three portions:

(1) Statement of scope, normative references, conformance requirements, and test methods (Section 1)

(2) Conventions and definitions (Section 2)

(3) Assertions to test POSIX.1 {3} interface facilities (Sections 2 through 10)

This introduction, any footnotes, notes accompanying the text, and the *informative* annexes are not considered part of this standard. Annexes A and B are informative.

Related Standards Activities

Activities to extend this standard to address additional requirements are in progress, and similar efforts can be anticipated in the future.

The following areas are under active consideration at this time, or are expected to become active in the near future:[1)]

(1) Language-independent service descriptions of POSIX.1 {3}

(2) C, Ada, and Fortran language bindings to (1)

(3) Verification testing methods

(4) Realtime facilities

(5) Secure/Trusted System considerations

(6) Network interface facilities

---

1) A *Standards Status Report* that lists all current IEEE Computer Society standards projects is available from the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903; Telephone: +1 202 371-0101; FAX: +1 202 728-9614.

Authorized licensed use limited to: Arizona State University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents
Document Sharing Hub

Full document details: IEEE 2003.1-1992

(7) System Administration

(8) Graphical User Interfaces

(9) Profiles describing application- or user-specific combinations of Open Systems standards for: supercomputing, multiprocessor, and batch extensions; transaction processing; realtime systems; and multiuser systems based on historical models

(10) An overall guide to POSIX-based or related Open Systems standards and profiles

Extensions are approved as "amendments" or "revisions" to this document, following IEEE and ISO/IEC procedures.

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

If you have an 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, and ask to have this forwarded to the chairperson of the appropriate TCOS working group. If you have interest in participating in this work at the international level, contact your ISO/IEC national body.

IEEE Std 2003.1-1992 was prepared by the P2003.1 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 P2003.1 working group was as follows:

Technical Committee on Operating Systems and Application Environments (TCOS)

Chair: Jehan-François Pâris

TCOS Standards Subcommittee

Chair:	Jim Isaak
Vice-Chairs:	Ralph Barker
	Hal Jespersen
	Lorraine C. Kevra
	Pete Meier
	Andrew T. Twigger
Treasurer:	Quin Hahn
Secretary:	Shane P. McCarron

P2003.1 Working Group Officials

Chair:	Roger J. Martin
Vice-Chairs:	Nicholas Ray Wilkes (1989-1990)
	Carol Raye (1987-1988)
Editor:	Anthony V. Cincotta
Secretaries:	Lowell G. Johnson (1989-1990)
	Doris Lebovits (1987-1988)

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

P2003.1 Technical Reviewers

Sanjay Agraharam	Mark Lamonds	Ravi Tavakley
Anthony V. Cincotta	James M. Moe	Andrew T. Twigger
Steve Henderson	Anita Mundkur	Bruce Weiner
	Gerald Powell	

P2003.1 Working Group

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

The following persons provided valuable input during the balloting period:

Chuck Karish	Paul Rabin

The following persons were members of the P2003.1 balloting group that approved this standard for submission to the IEEE Standards Board:

Mike Lambert	*X/Open Organizational Representative*
Shane P. McCarron	*Unix International Organizational Representative*
Fritz Schulz	*OSF Organizational Representative*

David Athersych	Judy Guist	Anita Mundkur
Geoff Baldwin	Charles E. Hammons	Martha Nalebuff
Jerome E. Banasik	Allen Hankinson	Barry Needham
Robert Barned	Craig Harmer	Fred Noz
Kabekode V. S. Bhat	Dale Harris	John C. Penney
Robert Bismuth	John L. Hill	P. J. Plauger
James Bohem	David F. Hinnant	Gerald Powell
Robert Borochoff	Lee A. Hollaar	Scott E. Preece
Keith Bostic	Terrence W. Holm	James M. Purtilo
James P. Bound	Richard Hughes-Rowlands	Carol Raye
Phyllis Eve Bregman	Jim Isaak	Christopher J. Riddick
A. Winsor Brown	Dan Iuster	Robert Sarr
Fred Lee Brown, Jr.	Scott Jameson	Norman Schneidewind
Nicholas A. Camillone	Hal Jespersen	Leonard W. Seagren
Clyde Camp	Lowell G. Johnson	Karen Sheaffer
John Caywood	Greg Jones	Dan Shia
Kilnam Chon	Lorraine C. Kevra	Mukesh Singhal
Chan F. Chong	Jeffrey S. Kimmel	Richard Sniderman
Anthony V. Cincotta	Martin J. Kirk	Douglas H. Steves
Robert L. Claeson	Dale L. Kirkland	Scott A. Sutter
Richard Cornelius	Kenneth C. Klingman	Robert Switzer
William M. Corwin	D. Richard Kuhn	Ravi Tavakley
Mike Cossey	Robin B. Lake	Donn S. Terry
William Cox	Mark Lamonds	Gary F. Tom
Donald W. Cragun	Doris Lebovits	Andrew T. Twigger
Ava Maria De Alvare	Robert M. Lenk	Mark-Rene Uchida
Terence Dowling	David Lennert	Michael W. Vannier
Stephen A. Dum	Kevin Lewis	John W. Walz
John D. Earls	Joseph F. P. Luhukay	Alan G. Weaver
Ron Elliott	Robert J. Makowski	Bruce Weiner
Philip W. Enslow	Roger J. Martin	Brian Weis

Authorized licensed use limited to: Wright State University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents
Document Sharing Hub

Full document details: IEEE 2003.1-1992

Kenneth T. Faubel	Joberto S. B. Martins	Peter J. Weyman
Terence Fong	Martin J. McGowan III	Andrew E. Wheeler, Jr.
Kenneth R. Gibb	Robert W. McWhirter	Nicholas Ray Wilkes
Michel Gien	Paul Merry	Oren Yuen
Gregory W. Goddard	Doug Michels	Alaa Zeineldine
Dave Grindeland	James M. Moe	Jason Zions

When the IEEE Standards Board approved this standard on December 3, 1992, it had the following membership:

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

Dennis Bodson	Donald N. Heirman	T. Don Michael*
Paul L. Borrill	Ben C. Johnson	John L. Rankine
Clyde Camp	Walter J. Karplus	Wallace S. Read
Donald C. Fleckenstein	Ivor N. Knight	Ronald H. Reimer
Jay Forster*	Joseph Koepfinger*	Gary S. Robinson
David F. Franklin	Irving Kolodny	Martin V. Schneider
Ramiro Garcia	D. N. "Jim" Logothetis	Terrance R. Whittemore
Thomas L. Hannan	Lawrence V. McCall	Donald W. Zipse

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
James Beall
Richard B. Engelman
David E. Soffrin
Stanley Warshaw

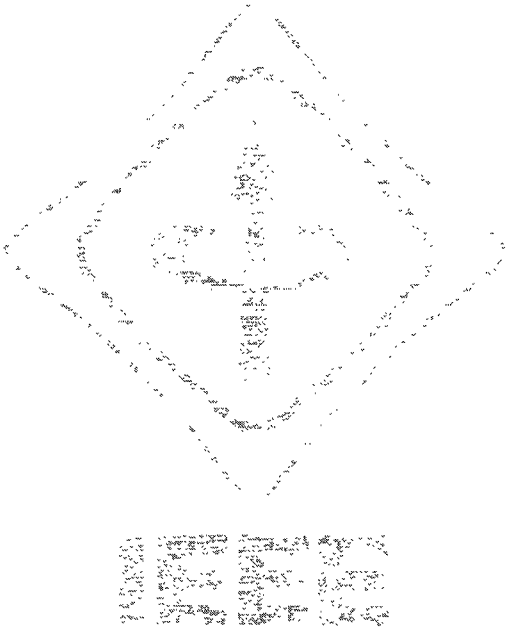
Mary Lynne Nielsen
IEEE Standards Project Editor

With the recommendation and endorsement of the standard developers, this standard is dedicated to the memory of Nicholas Ray Wilkes, who served as vice-chair of IEEE Std 1003.3-1991.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992



Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

IEEE Standard for Information Technology— Test Methods for Measuring Conformance to POSIX— Part 1: System Interfaces

Section 1: General

1.1 Scope

This standard defines the general requirements and test methods for measuring conformance to ISO/IEC 9945-1: 1990 (IEEE Std 1003.1-1990) hereinafter referred to as "POSIX.1" {3}.[1)] It also defines the test assertions for measuring conformance to POSIX.1 {3}.

This standard is intended for use by

— Developers of POSIX.1 {3} test methods;

— Implementors of POSIX.1 {3} implementations;

— Application writers for POSIX.1 {3} conforming implementations;

— POSIX.1 {3} testing laboratories; and

— Others interested in validating the conformance of a vendor-claimed POSIX.1 {3} implementation

The purpose of this standard is to specify the test assertions and related test methods for measuring conformance of an implementation to POSIX.1 {3}.

---

1) The numbers in curly braces correspond to those of the references in 1.2.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents Document Sharing Hub

Full document details: IEEE 2003.1-1992

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. These 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 POSIX.1 {3}; that normally would require exhaustive testing, which is impractical for both technical and economic reasons.

The technical specifications for a POSIX System Application Program Interface are defined in POSIX.1 {3}. IEEE Std 2003.1-1992 defines a means of measuring conformance to the POSIX.1 {3} technical specifications. Any question of interpretation of those technical specifications arising from the use of this standard is a question of interpretation of POSIX.1 {3}.

1.2 Normative References

The following standards contain provisions that, through references in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

{1} ISO/IEC 646: 1991,[2] *Information technology—ISO 7-bit coded character set for information interchange.*

{2} ISO/IEC 9899: 1990, *Information technology—Programming languages—C.*

{3} ISO/IEC 9945-1: 1990 (IEEE Std 1003.1-1990),[3] *Standard for Information Technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].*

{4} IEEE Std 1003.3-1991, *IEEE Standard for Information Technology—Test Methods for Measuring Conformance to POSIX.*

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

3) IEEE documents can be obtained from the The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, New York 10017, USA.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents
Document Sharing Hub

Full document details: IEEE 2003.1-1992

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Requirements

NOTE: The types of assertion numbering used in this document are explained in IEEE Std 1003.3-1991 {4}.

D01(A) The conformance document defines an environment in which an application can be run with the behavior specified by POSIX.1 {3} in 1.3.1.1 of the PCD.1[4)].

1.3.1.2 Documentation

D02(A) The conformance document contains a statement that indicates the full name, number, and date of the POSIX.1 {3} standard that applies in 1.3.1.2 of the PCD.1.

D03(C) If the implementation makes international software standards available to a conforming application and the implementation documents this:
The list of these standards is contained in 1.3.1.2 of the PCD.1.

NOTE: The conformance document requirements are specified in 1.4.17.1.

1.3.2 Application Conformance

There are no test methods provided for this subclause.

1.3.3 Language-Dependent Services for the C Programming Language

D04(C) If the C-language provided does not conform to the C Standard {2}:
The conformance documentation states that the C-language provided does not conform to the C Standard {2} in 1.3.3 of the PCD.1.

1.3.3.1 Types of Conformance

There are no test methods provided for this subclause.

1.3.3.2 C Standard Language-Dependent System Support

D05(C) If the C Standard {2} language binding is supported:
The conformance document states the version of the C Standard {2} referenced in fulfilling the requirements of Section 8 of POSIX.1 {3} in 1.3.3.2 of the PCD.1.

---

4) Abbreviations are defined in 1.4.2.3.

Documents Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

1.3.3.3 Common-Usage C Language-Dependent System Support

DGA01 For all functions stated in Section 8.1 of POSIX.1 {3}:
The details of all differences between the interface provided and the interface that would have been provided had the C Standard {2} been implemented instead of Common-Usage C are contained in 8.1 of the PCD.1.

D06(C) If the Common-Usage C language binding is supported and differences exist from the C Standard {2}:
The version of the C Standard {2} used for referencing the differences is stated in 1.3.3.3 of the PCD.1.

1.3.4 Other C Language-Related Specifications

NOTE: When a macro exists for any function, except *assert*(), *setjmp*(), and *sigsetjmp*(), then all assertions, except those in the Synopsis clause, are tested for both the macro and the function interfaces. The identifiers with external linkage providing access to *setjmp*() and *sigsetjmp*() shall be tested. Assertions in the Synopsis clause are specifically related to the function or macro interfaces as appropriate. (See 1.4.8.)

GA01 For all elements except *abort*(), *assert*(), *getc*(), *putc*(), *setjmp*(), *sigsetjmp*(), and *tzset*():
If the function is defined as a macro:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary.

For *funct*() of *setjmp*() and *sigsetjmp*():
If *funct*() is defined as a macro:
It protects its result value with extra parentheses when necessary.

There are no assertions specific to this subclause.

1.3.5 Other Language-Related Specifications

There are no test methods provided for this subclause.

1.3.6 Test Methods Conformance

1.3.6.1 Requirements

A conforming implementation of POSIX.3.1 test methods shall meet all of the following criteria:

(1) All test method requirements of 1.4.4 through 1.4.19 that are associated with required base assertions shall be implemented.

(2) If the test method tests for a specific POSIX.1 {3} profile, then all test method requirements of 1.4.4 through 1.4.19 that are associated with the requirements of this profile shall be implemented.

AVA Documents Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

(3) All required base assertions shall be tested.

(4) If the PCTS.1 tests for a specific POSIX.1 {3} profile, then all conditional base assertions associated with the requirements of this profile shall be implemented.

(5) A conforming PCTS.1 shall conform to the requirements of IEEE Std 1003.3-1991 (POSIX.3 {4}).

(6) A conforming PCTS.1 shall document the limitations on POSIX.1 {3} testing when employing the PCTS.1.

Example: This PCTS.1 considers appropriate privilege to be implemented monolithically as a single privilege.

1.3.6.2 Recommendations

A conforming implementation of POSIX.3.1 test methods should meet the following criterion:

— If extended assertions are within the POSIX.1 {3} profile being tested, then these assertions should, when possible, be tested.

1.4 Test Methods

1.4.1 Introduction

This clause defines the test methods to be used to measure conformance to POSIX.1 {3} and contains requirements and recommendations for the test methods specific to testing POSIX.1 {3}.

The contents of POSIX.3.1 are C-language specific because POSIX.1 {3} is specified in terms of C-language bindings. Except where explicitly stated, the requirements and recommendations of POSIX.3.1 apply to a PCTS.1 written in either Common-Usage C or C Standard {2} and test conformance to either Common-Usage C or C Standard {2} language bindings of a target system.

The test methods used for POSIX.1 {3} conformance measurement shall consist of a PCTS.1 and a documentation audit of the POSIX Conformance Document as required by POSIX.1 {3} and its associated PCTP.1.

1.4.2 Definitions

1.4.2.1 Terminology

1.4.2.1.1 may: An option for *test methods.*

In this standard the negative of *may* is *need not.*

1.4.2.1.2 shall: A requirement for *test methods.*

AVA Documents Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

1.4.2.1.3 should: A recommended practice for *test methods.*

1.4.2.2 General Terms

1.4.2.2.1 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. [IEEE Std 1003.3-1991]

1.4.2.2.2 assertion number: The numeric identifier assigned to an *assertion.*

The name of the *element* and the *assertion number* together uniquely identify an *assertion.* [IEEE Std 1003.3-1991]

1.4.2.2.3 assertion test: The software or procedural methods that ascertain the conformance of a POSIX implementation to an *assertion.* [IEEE Std 1003.3-1991]

1.4.2.2.4 base assertion: An *assertion* that is required to be tested for required features and for implemented *conditional features.* [IEEE Std 1003.3-1991]

1.4.2.2.5 conditional feature: A feature or behavior referred to in a POSIX standard that need not be present on all conforming implementations. [IEEE Std 1003.3-1991]

1.4.2.2.6 development system: The computer system used to compile and configure a PCTS.1. [IEEE Std 1003.3-1991]

1.4.2.2.7 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. [IEEE Std 1003.3-1991]

1.4.2.2.8 extended assertion: An *assertion* that is not required to be tested. [IEEE Std 1003.3-1991]

1.4.2.2.9 modem control: The monitoring of modem status lines.

1.4.2.2.10 POSIX Conformance Document (PCD): The conformance document required by a POSIX standard. [IEEE Std 1003.3-1991]

1.4.2.2.11 POSIX Conformance Test Procedure (PCTP): The nonsoftware procedures possibly used in conjunction with other test methods to measure conformance. [IEEE Std 1003.3-1991]

1.4.2.2.12 POSIX Conformance Test Suite (PCTS): The collection of software possibly used in conjunction with other test methods to measure conformance. [IEEE Std 1003.3-1991]

1.4.2.2.13 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. [IEEE Std 1003.3-1991]

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

1.4.2.2.14 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. [IEEE Std 1003.3-1991]

1.4.2.2.15 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. [IEEE Std 1003.3-1991]

1.4.2.2.16 test result code: A value that describes the result of an *assertion test.* [IEEE Std 1003.3-1991]

1.4.2.2.17 unattainable limit: A limit that is undefined for a *target system* or that has a magnitude exceeding the POSIX.1 {3} specified minimum and that would require an unreasonable amount of time or system resources to test.

1.4.2.3 Abbreviations

For the purposes of this standard, the following abbreviations apply:

1.4.2.3.1 C Standard: ISO/IEC 9899, *Information technology—Programming languages—C {2}.*

1.4.2.3.2 Common-Usage C: Common-Usage C Language-Dependent System Support.

1.4.2.3.3 PCD.1: POSIX Conformance Document for POSIX.1 {3}.

1.4.2.3.4 PCTP.1: POSIX Conformance Test Procedures for POSIX.1 {3}.

1.4.2.3.5 PCTS.1: POSIX Conformance Test Suite for POSIX.1 {3}.

1.4.2.3.6 POSIX: Colloquial name for the collection of IEEE 1003 standards, draft standards, and projects.

1.4.2.3.7 POSIX.1: ISO/IEC 9945-1: 1990 (IEEE Std 1003.1-1990).

1.4.2.3.8 POSIX.3: IEEE Std 1003.3-1991.

1.4.2.3.9 POSIX.3.1: The POSIX.3 standard that defines the test methods for the POSIX.1 {3} standard; a reference to this standard.

1.4.3 Assertion Concepts

1.4.3.1 Security Features

The assertions in this standard, in general, do not address the conditional features associated with extended security controls, alternate file access control mechanisms, and additional file access control mechanisms.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents Document Sharing Hub

Full document details: IEEE 2003.1-1992

1.4.3.2 Appropriate Privileges

All assertions that require either the presence or absence of appropriate privilege state their requirements. Assertions that contain no stated requirement relating to appropriate privilege hold true regardless of what appropriate privileges the calling process has.

1.4.3.3 Testing Constraints

A testing constraint is a conditional feature, behavior, or additional hardware or software needed to test an assertion. Testing constraints are employed within the classification of the assertion with the syntax:

(testing constraint?classification:UNTESTED)

1.4.3.4 PCD Announcement Mechanism

When assertions are dependent on the PCD.1 for an announcement mechanism about the support or nonsupport of POSIX.1 {3} conditional features, then the conditional phrase of the assertion shall incorporate the appropriate PCD symbol (see Table 1-4).

1.4.4 PCTS Execution

1.4.4.1 PCTS Target System Support Facilities

Support facilities are required to establish an operational environment for some tests within a PCTS.1, where it is necessary to undertake tasks that are beyond the scope of POSIX.1 {3}. The implementation of these support facilities will differ between different target systems, and it will be necessary for the user of the PCTS.1 to provide an interface to the support facilities to be able to execute the PCTS.1 fully.

1.4.4.1.1 Requirements

The interface syntax for a target system support facility shall be specified in the PCTS.1 documentation. The values designating successful and unsuccessful provision of the requested facility shall also be specified in the PCTS.1 documentation. A target system support facility shall return an unsuccessful indicator if the target system does not support the facility.

1.4.4.1.2 Recommendations

Where POSIX.1 {3} does not provide the means for establishing the required operational environment, a portable PCTS.1 should not make assumptions about the functions a target system will use to establish this environment. To establish the test environment, the PCTS.1 should use well-defined target system support facilities that may have to be provided by the user of the PCTS.1.

To increase portability, a PCTS.1 should allow the test installer to provide target system support facilities to establish an environment for testing.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

A target system support facility should be used to control

— Associating a controlling terminal with a process

— Establishing appropriate privileges for a process

— Mounting a file system with read and write capabilities

— Mounting a file system with read only capability

— Unmounting a file system

— Setting S_ISUID mode with *chmod*()

— Setting S_ISGID mode with *chmod*()

The PCTS.1 should not require information about the manner in which the implementation provides the requested facilities nor about the time taken to provide these facilities.

1.4.4.2 Test Tolerances

1.4.4.2.1 Requirements

In order to test for proper functioning of time-related functions, a PCTS.1 shall permit allowances for a degree of variability in the time to perform certain operations. For example, comparison of the times returned by the *times*() function to the times returned separately by the child processes will vary slightly between separate executions of the test.

POSIX.1 {3} does not define the timeliness or the asynchronous nature of certain events. The PCTS.1 shall expect that the effect of an event on both the initiating and all target processes shall be completed in a reasonable amount of time and that all of the affected processes shall continue in the expected manner after the event has occurred. The PCTS.1 shall report or document the time variation that it allows for a test tolerance.

1.4.4.2.2 Recommendations

None.

1.4.4.3 Timeout Tolerances

1.4.4.3.1 Requirements

In order to test certain time-related events, a PCTS.1 shall permit allowances for variance in the time at which the timer goes off. For instance, the *alarm*() function sets a timer that will cause a SIGALRM to be sent at the specified time. Variations in system load and process scheduling can cause the actual time at which SIGALRM is sent to vary.

1.4.4.3.2 Recommendations

A PCTS.1 should use a user configuration variable to specify the defined amount of time beyond the specified timer time that a test should wait before reporting failure.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

1.4.5 PCTS Testing Constraints

1.4.5.1 Requirements

The symbols to be used by a PCTS.1 when denoting POSIX.3.1 testing constraints are specified in Table 1-1.

Table 1-1 – Testing Constraints

Symbol	Testing Constraint
PCTS_APP_LINK_DIR	The implementation permits the appropriate privilege process to create links to directories.
PCTS_GTI_DEVICE	The implementation provides device types that support the general terminal interface.
PCTS_GTI_BUFFERS_OUTPUT	The general terminal interface device buffers its output.
PCTS_PROCESS_LIMIT	The system process limit is not exceeded before {CHILD_MAX} is reached.
PCTS_CHMOD_SET_IDS	The function *chmod*() allows setting the S_ISUID and S_ISGID modes.
PCTS_INVALID_SIGNAL	An invalid or unsupported signal number exists.
PCTS_INVALID_OWNER	An invalid group or owner ID exists.
PCTS_APP_MODE	The implementation permits the appropriate privilege process to change file mode.
PCTS_APP_OWNER	The implementation permits the appropriate privilege process to change owner and group IDs.
PCTS_APP_TIMES	The implementation permits the appropriate privilege process to change file times.
PCTS_ROOT_WRITABLE	The root file system is not read-only.
PCTS_HPA_FILE	The implementation supports the high-performance attribute file.

1.4.5.2 Recommendations

None.

1.4.6 PCTS Limits and Configuration Variables

1.4.6.1 Requirements

Table 1-2 defines sets of limits. The POSIX.1 {3} values {ARG_MAX}, {CHILD_MAX}, {LINK_MAX}, {MAX_CANON}, {MAX_INPUT}, {NAME_MAX}, {OPEN_MAX}, {PATH_MAX}, {PIPE_BUF}, and {TZNAME_MAX} used by the PCTS.1 shall be obtained from the *pathconf*() and *sysconf*() calls, unless the assertion explicitly states that the value is to be obtained from a specific header. The POSIX.1 {3} limit value for {PCTS_LOCK_MAX} is imposed by the system.

Because certain of these POSIX.1 {3} limits, for practical testing purposes, may be unattainable or require an unreasonable amount of time or resources before they are attained, values for the mimimum PCTS.1 testing limit (PCTS_mtl) are provided as specified in Table 1-2. When a specific PCTS_mtl is less than the associated POSIX.1 {3} limit, a PCTS.1 may set the actual test limit value (PCTS_tlv) such that PCTS_mtl is less than or equal to PCTS_tlv and PCTS_tlv is less than or equal to the POSIX.1 {3} limit. If the implementation does not provide a POSIX.1 {3} limit, the PCTS.1 shall set the test limit value such that PCTS_mtl is less than or equal to PCTS_tlv.

Documents Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

When the POSIX.1 {3} limit obtained from *pathconf*(), *sysconf*(), or another system-imposed limit is less than or equal to the associated PCTS_mtl, the POSIX.1 {3} limit shall be used for testing, as shown in Table 1-2.

Table 1-2 – Configuration Variables

PCTS Limit Name	POSIX.1 Limit	PCTS_mtl (PCTS.1 minimum test limit)
PCTS_ARG_MAX	{ARG_MAX}	The lesser of {ARG_MAX}, as obtained from *sysconf*(), and ten times {_POSIX_ARG_MAX}.
PCTS_CHILD_MAX	{CHILD_MAX}	The lesser of {CHILD_MAX}, as obtained from *sysconf*(), and 256.
PCTS_LINK_MAX	{LINK_MAX}	The lesser of {LINK_MAX}, as obtained from *pathconf*(), and 256.
PCTS_LOCK_MAX	system-imposed	The lesser of the system-imposed limit and 2500.
PCTS_MAX_CANON	{MAX_CANON}	The lesser of {MAX_CANON}, as obtained from *pathconf*(), and four times {_POSIX_MAX_CANON}.
PCTS_MAX_INPUT	{MAX_INPUT}	The lesser of {MAX_INPUT}, as obtained from *pathconf*(), and four times {_POSIX_MAX_INPUT}.
PCTS_NAME_MAX	{NAME_MAX}	The lesser of {NAME_MAX}, as obtained from *pathconf*(), and 2048.
PCTS_OPEN_MAX	{OPEN_MAX}	The lesser of {OPEN_MAX}, as obtained from *sysconf*(), and 256.
PCTS_PATH_MAX	{PATH_MAX}	The lesser of {PATH_MAX}, as obtained from *pathconf*(), and 4096.
PCTS_PIPE_BUF	{PIPE_BUF}	The lesser of {PIPE_BUF}, as obtained from *pathconf*(), and 32767.
PCTS_TZNAME_MAX	{TZNAME_MAX}	The lesser of {TZNAME_MAX}, as obtained from *sysconf*(), and 256.

The term *obtained from* in Table 1-2 refers to the meaning of the return value from a call to *pathconf*() or *sysconf*(). If this return value is -1 and the error indicator is unchanged, the PCTS.1 shall interpret this as a value greater than the corresponding PCTS_tlv.

The manner in which the assertion is modified is described in each of the assertions affected.

A PCTS.1 shall use configuration parameters to provide information about the control sequences used to perform specific terminal operations, as shown in Table 1-3. The sequences that an implementation uses to erase characters and lines differ from implementation to implementation. A PCTS.1 can assume no knowledge of these character sequences. The PCTS.1 will verify that the character sequence used to erase the character/line matches the expected sequence for the character/line erase operation for the target system. These character sequences should be provided to the PCTS.1 using the configuration variables specified in Table 1-3.

Table 1-3 – Configuration Control Sequences

Test Option	Value
PCTS_ECHOE	Erase character sequence, or empty if none.
PCTS_ECHOK	Erase line sequence, or empty if none.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents
Document Sharing Hub

Full document details: IEEE 2003.1-1992

The testing of character and line erase sequences is undertaken in canonical mode. The PCTS.1 shall set up the test for character erasure by first ensuring that the terminal input line is empty and then placing a sequence of characters into the input line. The PCTS.1 shall then issue an erase character command and ensure that the sequence specified in {PCTS_ECHOE} is used to erase the character. A similar technique is used to test for line erasure except the sequence specified in {PCTS_ECHOK} is used to check that the correct sequence of characters is used to erase the line.

1.4.6.1.1 Recommendations

None.

1.4.7 Test Methods for Headers

1.4.7.1 Requirements

The symbol _POSIX_SOURCE shall be defined by the PCTS.1 prior to the inclusion of any headers.

Since headers are not required to be readable files of C source code on the target system, it is necessary to use indirect ways to test their content. The items defined in headers shall be tested as follows to establish compliance as required in POSIX.1 {3}.

— *Symbolic constants, macros*, and *types* shall be tested for existence and their functional correctness. See 1.4.8 for more requirements on macros.

— *Function prototypes* in the C Standard {2} shall be tested for their existence and correctness of definition.

— *Function declarations* in Common-Usage C shall be tested for their existence and correctness of definition.

— *Function return values* in Common-Usage C for functions not returning integer type shall be tested for correctness of return value.

— *Data aggregate definitions* shall be tested for their existence and correctness of all members and types associated with members that are specified in POSIX.1 {3}.

— *Variables* shall be tested for their existence and types.

— The ability to include a header more than once shall be tested.

— The ability to access a symbol whose definition is required in more than one header shall be tested.

1.4.7.2 Recommendations

Headers should be tested by including them in test programs. The following criteria should be used for type checking.

— For a definition of a structure type, the size of each structure member should be compared to the size of an item of the type of the member or to the size of the type of the member itself.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents
Document Sharing Hub

Full document details: IEEE 2003.1-1992

- For a variable declaration, the size of the variable should be compared to the size of a variable of the specified type or to the size of the specified type itself. This test applies to members of structures, not to structures themselves.
- For a variable, the variable should be compared to another variable of the same type. This test applies to members of a structure, not to structure variables.
- For a variable, the variable should be assigned to another variable of the same type.
- For a variable, the address of the variable should be assigned to another variable of the type pointer to the defined type. This does not apply to function return values.

The detection of a nonconformity relies upon the C compiler generating an error or warning message. If such a message is generated, then there should be either a PCTP.1 to check the compiler output to resolve whether the tests **PASS** or **FAIL**, or software to do the checking automatically.

To test for proper behavior of duplicate header inclusion and duplicate symbolic definition in headers, a test should include all of the headers to be tested in a valid order such that there are duplicate inclusions of all headers.

If the PCTS.1 uses a language compiler or translator to accomplish testing of the assertions related to headers, the PCTS.1 should not rely solely on the return value from that compiler or translator, but should also examine any messages produced during test execution.

1.4.8 Test Methods for Macros

1.4.8.1 Requirements

A PCTS.1 shall test that any interface that is implemented as a macro has an underlying function of the same name (if required by the standard) and that both the macro and the function behave in the same manner with respect to the assertions.

The PCTS.1 shall also test that the macro is implemented in such a manner that it evaluates its arguments only once, fully protected by parentheses when necessary, and yields a resulting value that can be used in complex expressions without requiring extra parentheses.

1.4.8.2 Recommendations

None.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents
Document Sharing Hub

Full document details: IEEE 2003.1-1992

1.4.9 Test Methods for Invalid Constant Values

1.4.9.1 Requirements

Some POSIX.1 {3} functions take a symbolic constant as an argument and have a specified behavior when the argument has an invalid value. Because an implementation can have extensions that permit the function to accept values other than those specified by POSIX.1 {3}, a portable PCTS.1 shall not simply choose such a value.

1.4.9.2 Recommendations

A PCTS.1 should use a user configuration variable to select an invalid value of each constant for which the behavior of invalid values is tested.

1.4.10 Test Methods for *errno* Checking

1.4.10.1 Requirements

A test shall be performed for detection of all values of *errno* specified in the Errors subclause of each element. If more than one error occurs in processing a function call, an assertion test shall not depend on the order in which the errors are detected or in which error code is returned.

Tests for error code detection shall be coded to generate only a single error code when possible.

1.4.10.2 Recommendations

None.

1.4.11 Test Methods for File Accessibility

1.4.11.1 Requirements

POSIX.1 {3} defines the cases in which access is granted to a file from a POSIX.1 {3} interface both in the case that the implementation provides a mechanism for a process to exist with the appropriate privilege to override the standard file access control mechanism and the case that the process has standard file access capabilities. A PCTS.1 shall test for the accessibility of a file in both of these cases and shall test for the [EACCES] error indication in the case that standard file access is denied to a process. These tests shall be carried out for each of the interfaces that contain assertions refering to the ability to access a file or assertions refering to the [EACCES] error.

POSIX.1 {3} defines the cases of additional and alternate file access control mechanisms without defining the manner in which these mechanisms can be activated. A PCTS.1 is not required to test these mechanisms, but shall run in an environment where these mechanisms exist but are not enabled.

AVA Documents Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

1.4.11.1.1 Recommendations

None.

1.4.12 Test Methods for Signal Functions

1.4.12.1 Reentrance of Interrupted Functions

1.4.12.1.1 Requirements

A PCTS.1 shall test that, when a function in a quiescent state is interrupted by a signal, a call to the interrupted function from the signal handling routine behaves correctly and does not improperly affect the result of the interrupted function.

1.4.12.1.2 Recommendations

None.

1.4.12.2 Functions Interruptable by Signals

1.4.12.2.1 Requirements

When executing a test to verify that *errno* is set to [EINTR] when a function is interrupted, a PCTS.1 shall not use the default action of job control signals to effect the interruption. These signals cause the process to stop and do not actually interrupt the function.

1.4.12.2.2 Recommendations

None.

1.4.12.3 *sigaction*() Testing

1.4.12.3.1 Requirements

See *sigaction*() assertions.

1.4.12.3.2 Recommendations

If the behavior associated with {_POSIX_JOB_CONTROL} is supported, the recommended test method is to establish a known value for a globally accessible variable and then to call *sigaction*() for the SIGSTOP signal to attempt to associate a signal catching function with this signal. The handler would be written to change the known value. A SIGSTOP signal should then be generated, followed by a SIGCONT signal to cause the process to continue. If the known value has changed, the handler was established even though *sigaction*() failed.

If the behavior associated with {_POSIX_JOB_CONTROL} is not supported, the recommended method for testing that the signal catching function is not associated with the signal when *sigaction*() fails is to install a signal-catching function for SIGKILL that changes the value of a global variable. Then a SIGKILL signal is sent to the process. If the process can check whether the global variable has been

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents
Document Sharing Hub

Full document details: IEEE 2003.1-1992

changed (even determining that it has not), SIGKILL has not been delivered properly.

Possible tests include

— Having the handler do an *_exit*(*n*) and having the parent check exit status

— Writing to the file system from the handler

1.4.13 Test Methods for Files and Input/Output Primitives

1.4.13.1 Requirements

A PCTS.1 shall use at least one of the file types that are defined in POSIX.1 {3} when testing an assertion that applies to files. The file type chosen may differ for different assertions. Wherever a specific set of file types are defined in an assertion, a PCTS.1 shall use each of the file types stated.

1.4.13.2 Recommendations

A PCTS.1 implementor may choose to improve test coverage by increasing the number of file types covered by assertions.

1.4.14 File Descriptor Closure Test Method

1.4.14.1 Requirements

A PCTS.1 shall test that file descriptors are closed by *_exit*(), *exit*(), and *abort*(). (See GA41 in 3.2.)

1.4.14.2 Recommendations

The recommended method for testing file descriptor closure on a regular file is as follows:

— Parent process opens a file, then creates a child with *fork*().

— The child process creates an exclusive lock on the file and calls *_exit*().

— Upon termination of its child process, the parent attempts to exclusively lock the same part of the file. If it can, the test passes. If it cannot, the test fails.

The recommended method for testing this on a pipe (or FIFO) is as follows:

(1) Create a pipe.

(2) Fork a child process.

(3) Close the write channel of the pipe in the parent process.

(4) Terminate the child process.

(5) Expect a *read*() on the pipe in the parent process to return a value of 0.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

1.4.15 Test Methods for Terminal I/O Functions

1.4.15.1 Requirements

Terminal I/O testing shall not be performed using an interactive test method, and the implementation shall provide a method whereby a process can acquire a controlling terminal.

1.4.15.2 Recommendations

Terminal I/O should be tested using an automated test method. This method provides for accuracy and repeatability of test results by eliminating the need for human intervention while running a PCTS.1. An automated method of terminal I/O testing requires two asynchronous ports to be electrically connected in closed-loop mode and for these ports to support modem control protocols. One of the closed-loop ports is the driver port and the other is the target port. The terminal I/O tests send character data between the driver port and the target port via the connecting cable as though the driver port were a terminal. The tests then compare the data received with the expected results.

Target systems normally configured with less than two asynchronous ports may be specifically configured with two asynchronous ports for the terminal I/O tests. The normal configuration will be considered to have passed the terminal I/O tests if the special two-asynchronous-port configuration has passed the tests.

A more limited method that may be used if only one asynchronous port is available is to wire the port to loop the output of the port back to the input of the same port.

A PCTS.1 may determine the attributes of an asynchronous port through the use of the *tcsetattr*() and *tcgetattr*() functions. An attribute, such as CLOCAL for modem control, may be set using *tcsetattr*(). If that value is not set after being set by *tcsetattr*() and read by *tcgetattr*(), then modem control is not supported by that serial port.

1.4.16 Test Methods for File Formats

1.4.16.1 Requirements

The PCTS.1 shall supply all files used to test file formats (see Section 10). These file formats are tested using the file reading utility. Do not create them on the target system with the file creating utility.

Where the PCTS.1 tests the content of text in file format headers, it shall do so using the ISO/IEC 646:1991 {4} IRV character representation (see Section 10).

1.4.16.2 Recommendations

None.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents Document Sharing Hub

Full document details: IEEE 2003.1-1992

1.4.17 Documentation Audit

1.4.17.1 Requirements

The purpose of the documentation audit is to verify that the PCD.1 meets the requirements specified in Section 1.3.1.2 of POSIX.1 {3} as enumerated by the documentation assertions. These general requirements are that the PCD.1

- Has the same structure as POSIX.1 {3}, with the information presented in the appropriately numbered sections, clauses, and subclauses. Information may appear under a higher level heading or in a section, clause, or subclause whose content in POSIX.1 {3} is intended to cover multiple sections, clauses, or subclauses throughout the standard.
- Does not contain information about extended facilities or capabilities outside the scope of POSIX.1 {3} and only contains information related to the statements in Section 1.3.1.2 of POSIX.1 {3}
- Describes the limit values found in the `<limits.h>` and `<unistd.h>` headers and listed in Sections 2.8 and 2.9 of POSIX.1 {3}, stating values, the conditions under which those values may change, and the limits of such variations, if any
- Describes the behavior of the implementation for all implementation-defined features defined in POSIX.1 {3}
- May specify the behavior of the implementation for those features where POSIX.1 {3} states that implementations may vary or implies variations among implementations
- May specify the behavior of the implementation for those features where features are identified as undefined or unspecified, or where the POSIX.1 {3} text implies these terms
- Specifies all items of POSIX.1 {3} where the phrases "shall document" or "shall be documented" are used

Extended security controls may necessitate features to be documented that do not have explicit documentation assertions. Such additional documentation shall be allowed as long as it is revelant to features specified in POSIX.1 {3}.

The test result code appropriate for each documentation assertion shall be determined. When the information specified by a documentation assertion does not appear in the designated section or sections, the result code for that assertion shall be **FAIL**.

Conformance documents, required by standard organizations, may be bound together as long as the POSIX.1 {3} part is clearly identifiable.

1.4.17.2 Recommendations

None.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

1.4.18 PCD Announcement Mechanism

1.4.18.1 Requirements

When assertions are dependent on the PCD.1 for an announcement mechanism about the support or nonsupport of POSIX.1 {3} conditional features, the symbol to be used and its associated conditional feature are stated in Table 1-4.

Table 1-4 – PCD Determinable Conditional Features

Symbol	Documentation
{PCD_CREAT_LINK_COUNT}	If supported, then creating a directory causes the link count of the directory in which it is created to be incremented.
{PCD_DIR_TYPE}	If supported, then type DIR is implemented as a file descriptor.
{PCD_LINK_TO_DIRECTORY}	If supported, then a calling process with the proper permission can create a link to a directory.
{PCD_LINK_FILE_SYSTEM}	If supported, then a link can be created between different file systems.
{PCD_NO_LOCK_FILE_TYPE}	If supported, then file types for which locking is unsupported do exist.
{PCD_READ_INTERRUPTED}	If supported, then *read*() when interupted by a signal after successfully reading some data returns the number of bytes read, otherwise returns an error value.
{PCD_WRITE_INTERRUPTED}	If supported, then *write*() when interupted by a signal after successfully writing some data returns number of bytes written, otherwise returns an error value.
{PCD_WRITE_PERM_TO_RENAME}	If supported, then write permission is required for an existing directory to rename it.

1.4.18.2 Recommendations

None.

1.4.19 Test Methods for Common-Usage C Functions

1.4.19.1 Requirements

If an implementation claiming conformance to Common-Usage C does not support a C Standard {2} function or a PCTS.1 detects a deviation from the C Standard {2}, the corresponding assertions in 8.1.1 through 8.1.59 shall have the result code **UNRESOLVED** or **FAIL**. If the audit of the PCD.1 shows that the PCD.1 declares

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents
Document Sharing Hub

Full document details: IEEE 2003.1-1992

the C function not implemented or the deviations are documented, the test result code is resolved to **PASS**; otherwise it remains **FAIL.** Additionally, if differences are not documented in the PCD.1, documentation assertion "D01" in Section 8 shall have the result code **FAIL.**

This means that effectively each assertion in 8.1.1 through 8.1.59 (although not written that way) reads as follows:

XX(A) If the function is supported and there is no deviation from (assertion):
(assertion)

Otherwise:
Either the PCD.1 declares the C function not implemented, or the PCD.1 documents the deviation.

1.4.19.2 Recommendations

None.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents
Document Sharing Hub
Full document details: IEEE 2003.1-1992

Section 2: Definitions and General Requirements

2.1 Conventions

This standard uses the following typographic conventions:

(1) IEEE Std 1003.3-1991 {4} defines the nomenclature required for writing assertions. The description of the terms (A), (B), (C), (D), (Condition?A:B), R, GA, and the testing requirements are discussed in that standard.

(2) When assertions are dependent on testing constraints, the nomenclature as described in 1.4.3.3 is used.

(3) When assertions are dependent on PCD conditionals, the nomenclature as described in 1.4.3.4 is used.

(4) Assertion lists generated for each POSIX.1 {3} element begin with a new clause or subclause. Each clause and subsequent subclauses contain a trailing POSIX.1 {3} subclause reference in parentheses.

(5) The *italic* font is used for

— The initial appearances of defined terms

— Cross-references to defined terms within 1.4.2

— 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

— The trailing POSIX.1 {3} subclause reference of the element

(6) The **bold** font is used for

— Test result codes

(7) 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, element groupings, and C-language headers

(8) Symbolic *errno* names returned by functions are represented as [Symbolic_name].

(9) Symbolic constant limits are represented as {Symbolic_constant_limit}.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

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

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

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

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

(14) 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.

(15) The double quote ("name") is used when a name is specified.

(16) The single quote ('value') is used when a specific value is specified.

(17) 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 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	`tar`
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	long int
Function name	*funct*()
Global external	*errno*
IEEE Std 1003.n-19xy reference	POSIX.n
Implementation-dependent limit	{MAX_INPUT}
Metacharacter	* (any character string)
Null test method	UNUSED
Parameters	*<path>*
Section x.y reference	See x.y or x.y
Special character	`<newline>`
Symbolic constant limit	{LINK_MAX}
Symbolic constant option	{_POSIX_JOB_CONTROL}
Symbolic constant value	{_POSIX_JOB_CONTROL}
Symbol defined in header	_POSIX_JOB_CONTROL
Table reference	Table 2-1
Variable	*st_atime*

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2010 at 01:31:01 EST from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

2.2 Definitions

2.2.1 Terminology

There are no test methods provided for this subclause.

2.2.2 General Terms

NOTE: The subclause numbers that follow are not sequential because not all definitions have associated test assertions. Only those with specific test assertions are presented here.

2.2.2.4 Appropriate Privileges

D01(A) The means for associating the appropriate privileges with a process are described in 2.2.2.4 of the PCD.1 or in all the subclauses of the PCD.1 that correspond to subclauses in POSIX.1 {3} where privileged behavior is described.

2.2.2.9 Character Special File

D02(C) If character special files other than terminal device files are supported, and this is documented:
The details are contained in 2.2.2.9 of the PCD.1.

D03(C) If structures of character special files other than terminal device files are described:
The details are contained in 2.2.2.9 of the PCD.1.

2.2.2.27 File

D04(C) If file types other than regular file, character special file, block special file, FIFO special file, and directory are described:
The details are contained in 2.2.2.27 of the PCD.1.

2.2.2.30 File Group Class

D05(C) If the implementation uses additional criteria beyond those specified by POSIX.1 {3} to assign a process to the file group class of a file, and this is documented:
The details are contained in 2.2.2.30 of the PCD.1.

2.2.2.32 Filename

GA02 For *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), *remove*(), `tar` format creating utility, and `cpio` format creating utility:
An implementation supports *filenames* containing any of the following characters:

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Documents Document Sharing Hub
Full document details: IEEE 2003.1-1992

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

GA03 For *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), *remove*(), `tar` format creating utility, and `cpio` format creating utility:

A *filename* retains the unique identity of its upper- and lowercase letters.

01(B) No two directory entries in the same directory have the same name.

See Reason 3 in Section 5 of POSIX.3 {4}.

2.2.2.55 Parent Process ID

D06(A) The new parent process ID assigned to a process after the lifetime of its parent has ended is stated in 2.2.2.55 of the PCD.1.

2.2.2.57 Pathname

D07(A) Any special interpretation of a *pathname* that begins with exactly two slashes is stated in 2.2.2.57 of the PCD.1.

2.2.2.64 Process Group ID

02(B) A process group ID is not reused by the system until the process group lifetime ends.

See Reason 3 in Section 5 of POSIX.3 {4}.

2.2.2.66 Process Group Lifetime

03(A) The lifetime of the process group ends when the last remaining process leaves the group.

Testing Requirements:

Test that the process group for the last remaining process in the process group that has been terminated but unwaited for still exists.

2.2.2.67 Process ID

04(B) A process that is not a system process does not have a process ID of 1.

See Reason 3 in Section 5 of POSIX.3 {4}.

05(B) A process ID is not reused by the system until the process lifetime ends.

See Reason 3 in Section 5 of POSIX.3 {4}.

06(B) A process ID is not reused while there exists a process group with a process group ID of the same number.

See Reason 3 in Section 5 of POSIX.3 {4}.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

2.2.2.68 Process Lifetime

D08(C) If documentation is provided that states the resources returned when a process terminates:

The details are contained in 2.2.2.68 of the PCD.1.

07(B) When a process completes a *wait*() or *waitpid*() for a process that is inactive due to termination, then all of the resources associated with the inactive process are returned to the system. The last of the resources returned is the process ID.

See Reason 3 in Section 5 of POSIX.3 {4}.

2.2.2.69 Read-Only File System

D09(A) The specific restrictions on the semantics of system interfaces as they apply to objects on read-only file systems are contained in 2.2.2.69 of the PCD.1.

2.2.2.83 Supplementary Group ID

D10(C) If the effective group ID of a process is included in the list of supplementary group IDs returned by *getgroups*(), and this is documented:

The details are contained in 2.2.2.83 of the PCD.1.

2.3 General Concepts

2.3.1 Extended Security Controls

D01(C) If extended security controls are supported, and this is documented:

The extended security controls supported are stated in 2.3.1 of the PCD.1.

01(D) If the implementation supports extended security controls:

Extended security controls do not alter or override the defined semantics of any of the functions in POSIX.1 {3}.

See Reason 1 in Section 5 of POSIX.3 {4}.

2.3.2 File Access Permissions

D02(C) If whether additional or alternate file access control mechanisms are supported is documented:

The manner of support for additional or alternate file access control mechanisms or information that they are not supported is contained in 2.3.2 of the PCD.1.

GA04 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), and *pathconf*():

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

If the implementation provides the mechanism for creating processes with the appropriate privilege to override a file access control mechanism:

When the process has appropriate privileges for file access, and a call to function *funct*() requires read, write, or search access to the *pathname* argument, then the process is granted access to the file.

GA05 For *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), and *execvp*():

If the implementation provides the mechanism for creating processes with the appropriate privilege to override a file access control mechanism:

When the process has the appropriate privileges for file access, execute permission is granted to at least one user of the file, and a call to an exec type function requires execute access to the argument *path* or *file*, then the process is granted access.

GA06 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), and *pathconf*():

When the process does not have the required appropriate privileges to override the file access control mechanism, and the process requires read, write, execute, or search access to the pathname argument of function *funct*(), then a call to *funct*() is granted access in the case that the required access permission bit is set for the class (file owner class, file group class, or file other class) to which the process belongs.

GA07 If the implementation supports additional file access control mechanisms:

The mechanism only further restricts the access permissions defined by the file permission bits.

GA08 If the implementation supports alternate file access control mechanisms:

The mechanism specifies file permission bits for the file owner class, file group class, and file other class of the file corresponding to the access permissions to be returned by *stat*() and *fstat*().

GA09 If the implementation supports alternate file access control mechanisms:

The mechanism can be enabled only by explicit user action on a per-file basis by a process having the required appropriate privilege.

GA10 If the implementation supports alternate file access control mechanisms:

The mechanism can be enabled only by explicit user action on a per-file basis by the file owner.

GA11 If the implementation supports alternate file access control mechanisms:

The mechanism can be disabled for a file after the file permission bits are changed for that file with *chmod*().

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 22,2024 at 03:03:03 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

D03(C) If both alternate and additional file access control mechanism(s) are provided, and whether disabling an alternate mechanism after *chmod*() disables additional mechanisms is documented:
The details are contained in 2.3.2 of the PCD.1.

2.3.3 File Hierarchy

There are no test methods provided for this subclause.

2.3.4 Filename Portability

There are no test methods provided for this subclause.

2.3.5 File Times Update

R01 For *close*():
When a file is closed by the last process that had it open, then all time-related fields marked for update are updated. [See Assertion 11 in 6.3.1.2.]

GA12 For *stat*() and *fstat*():
When *stat*() or *fstat*() is called for a file, then all time-related fields marked for update are updated, and time-related fields not marked for update are not updated.

D04(C) If an implementation of a function that is required by POSIX.1 {3} to update time-related fields for reasons other than pathname resolution updates fields other than those required:
The details are contained in 2.3.5 of the PCD.1 or in the subclause of the PCD.1 that corresponds to the *description* subclause in POSIX.1 {3} where the function is defined.

D05(C) If an implementation of a function that is not required by POSIX.1 {3} to update time-related fields happens to do so, and this is documented:
The details are contained in 2.3.5 of the PCD.1 or in the subclause of the PCD.1 that corresponds to the *description* subclause in POSIX.1 {3} where the function is defined.

D06(C) If whether the updating of the related fields is done immediately or periodically is documented:
The details are contained in 2.3.5 of the PCD.1 or in the subclause of the PCD.1 that corresponds to the *description* subclause in POSIX.1 {3} where the function is defined.

GA13 For *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *access*(), *chmod*(), *chown*(), and *utime*():
No time-related fields are updated on read-only file systems.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Documents Document Sharing Hub
Full document details: IEEE 2003.1-1992

2.3.6 Pathname Resolution

NOTE: In each of the pathname resolution general assertions below, for the elements *rmdir*(), *rename*() new, and *unlink*(), the current working directory should be an empty directory in order to avoid the occurrence of avoidable error conditions. Some implementations will consider the attempt to remove the current working directory an error and will indicate this with the [EBUSY] error indication.

GA14 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():
When the first filename component of the pathname argument is ".", and the pathname does not begin with a slash, then *funct*() resolves the pathname by locating the second filename component (when specified) in the current working directory.

GA15 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():
When the argument *path* or *file* points to the string "/", then *funct*() resolves the pathname to the root directory of the process.

GA16 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():
When the argument *path* or *file* points to the string "///", then *funct*() resolves the pathname to the root directory of the process.

GA17 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():
When the pathname argument points to a string beginning with a single slash or beginning with three or more slashes, then *funct*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process.

GA18 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():
When the first filename component of the pathname argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *funct*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory.

GA19 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

AVA Documents Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2025 at 08:49:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

When the argument *path* or *file* points to the string "`F1/`" and `F1` is a directory, then *funct*() resolves the pathname to `F1`, which is in the current working directory.

GA20 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the argument *path* or *file* points to the string "`F1//`" and `F1` is a directory, then *funct*() resolves the pathname to `F1`, which is in the current working directory.

For *rename*() new:

When the argument *new* points to the string "`F1//`" and `F1` is an empty directory, then *rename*() resolves the pathname to `F1`, which is in the current working directory.

GA21 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the pathname argument points to the string "`F1/F2`", then *funct*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory.

GA22 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the pathname argument points to the string "`F1/./F2`" , then *funct*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory.

GA23 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the pathname argument points to the string "`F1/../F1/F2`", then *funct*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory.

GA24 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the pathname argument points to the string "`F1//F2`", then *funct*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory.

GA25 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Documents
Document Sharing Hub

Full document details: IEEE 2003.1-1992

When the pathname component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *funct*() resolves the pathname component by truncating it to {NAME_MAX} bytes.

There are no assertions specific to this subclause.

2.4 Error Numbers

01(A) The element *errno* is defined and available to a process via

extern int errno;

02(A) When the header <errno.h> is included, then the error numbers [E2BIG], [EACCES], [EAGAIN], [EBADF], [EBUSY], [ECHILD], [EDEADLK], [EDOM], [EEXIST], [EFAULT], [EFBIG], [EINTR], [EINVAL], [EIO], [EISDIR], [EMFILE], [EMLINK], [ENAMETOOLONG], [ENFILE], [ENODEV], [ENOENT], [ENOEXEC], [ENOLCK], [ENOMEM], [ENOSPC], [ENOSYS], [ENOTDIR], [ENOTEMPTY], [ENOTTY], [ENXIO], [EPERM], [EPIPE], [ERANGE], [EROFS], [ESPIPE], [ESRCH], and [EXDEV] are defined, are nonzero, are distinct from each other, and can be represented in *errno*.

03(B) No function defined by POSIX.1 sets *errno* to zero to indicate an error.

See Reason 2 in Section 5 of POSIX.3 [4].

D01(C) If the implementation supports additional errors not listed in 2.4, and this is documented:

The details are contained in 2.4 of the PCD.1 or in the subclause of the PCD.1 that corresponds to the *errors* subclause in POSIX.1 [3] for each interface that returns the error.

DGA02 For *fork*(), *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*(), *sigaddset*(), *sigdelset*(), *sigismember*(), *opendir*(), *readdir*(), *closedir*(), *getcwd*(), *access*(), *chown*(), *pathconf*(), *fpathconf*(), and *fcntl*():

If the implementation supports the detection of optional error conditions:

The details of the error conditions detected are contained in the subclause of the PCD.1 where the error values of the function are described.

GA26 For *fork*(), *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *sigaddset*(), *sigdelset*(), *sigismember*(), *opendir*(), *readdir*(), *closedir*(), *access*(), *chown*(), and *fcntl*():

If the implementation does not support the detection of optional error conditions:

The associated action requested is successful (unless a different error condition is detected).

D02(A) The details concerning whether [EFAULT] is reliably detected are contained in 2.4 of the PCD.1 or in the subclause of the PCD.1 that corresponds to the *errors* subclause in POSIX.1 [3] for each interface that is documented optionally to return [EFAULT].

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 23,2021 at 07:08:11 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

D03(A) The maximum file size allowed by the implementation is contained in 2.4 of the PCD.1.

2.5 Primitive System Data Types

01(A) When the header <sys/types.h> is included, then the data types *dev_t*, *gid_t*, *ino_t*, *mode_t*, *nlink_t*, *off_t*, *pid_t*, *size_t*, *ssize_t*, and *uid_t* are defined and are arithmetic data types.

02(A) The data type of each of *pid_t*, *off_t*, and *ssize_t* is signed arithmetic.

03(A) The data type of *size_t* is unsigned arithmetic.

04(A) The type *ssize_t* is capable of storing values in a range from "–1" to SSIZE_MAX.

05(B) When the header <unistd.h> is included, then the data types *size_t* and *ssize_t* are defined.

See Reason 1 in Section 5 of POSIX.3 {4}.

D01(C) If additional type symbols ending in "_t" are defined in any header specified by POSIX.1 {3}, and this is documented:
The details are contained in 2.5 of the PCD.1.

2.6 Environment Description

01(C) If the environment variable **HOME** was defined by the implementation and currently has the value defined by the implementation:
The environment variable **HOME** corresponds to the *initial working directory* of the user from the user database.

02(D) If the environment variable **LOGNAME** was defined by the implementation and currently has the value defined by the implementation:
The environment variable **LOGNAME** corresponds to the *login name* associated with the current process.

NOTE: Testing that **LOGNAME** is composed of *characters from the portable filename character set* is not asserted since POSIX.1 {3} implies in a note that this condition should be tolerated.

See Reason 2 in Section 5 of POSIX.3 {4}.

GA27 For *execlp*() and *execvp*():
Only when the argument *file* does not contain a slash are the **PATH** prefixes used by *execlp*() and *execvp*().

GA28 For *execlp*() and *execvp*():
A colon ':' in **PATH** separates one path prefix from another. Each path prefix in **PATH** indicates a directory to be included in the search path.

GA29 For *execlp*() and *execvp*():
When a nonzero-length prefix is applied to a filename, then a '/' is inserted between the prefix and the filename.

Authorized licensed use limited to: Carnegie Mellon University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Documents Document Sharing Hub
Full document details: IEEE 2003.1-1992

GA30 For *execlp*() and *execvp*():
An occurrence of two adjacent colons '::' in **PATH** indicates that the current working directory is to be included in the search path.

GA31 For *execlp*() and *execvp*():
An initial ':' indicates the current working directory.

GA32 For *execlp*() and *execvp*():
A trailing ':' indicates the current working directory.

GA33 For *execlp*() and *execvp*():
The directories in the search path are searched in the order in which they occur in the **PATH** environment until an executable file of the specified name is found.

GA34 For *getenv*() and `exec` type functions:
Upper- and lowercase letters in the environment retain their unique identities.

GA35 For *getenv*() and `exec` type functions:
An implementation supports environment variable names consisting of characters in the portable filename character set.

D01(C) If the implementation supports the use of characters other than those in the portable character set in environment variable names, and this is documented:
The details are contained in 2.6 of the PCD.1.

2.7 C Language Definitions

2.7.1 Symbols From the C Standard

01(A) When the header `<time.h>` is included, then the types *clock_t* and *time_t* and the macro NULL are defined.

02(A) The type *clock_t* is capable of representing all integer values from zero to the number of clock ticks in 24 h.

03(A) When the header `<unistd.h>` is included, then the symbol NULL is defined and expands to either an integral constant expression with the value 0 or such an expression cast to type *(void *)*.

04(A) Each of the following headers `<dirent.h>`, `<fcntl.h>`, `<grp.h>`, `<pwd.h>`, `<setjmp.h>`, `<signal.h>`, `<stdio.h>`, `<sys/stat.h>`, `<sys/times.h>`, `<sys/wait.h>`, `<termios.h>`, `<time.h>`, `<utime.h>`, and `<unistd.h>` can be included more than once, in any combination of headers in any order, and a symbol can be defined with the same value in more than one of these headers.

NOTE: It is possible to include `<sys/types.h>` more than once, and the first instance of its inclusion precedes any other header that depends upon its prior inclusion.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2023 at 07:37:37 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

2.7.2 POSIX.1 Symbols

NOTE: When {_POSIX_SOURCE} is defined before any header is included, then those symbols defined by POSIX.1 {3} to be in the header are visible. (See 1.4.7.1.)

05(B) When {_POSIX_SOURCE} is defined and no other feature test macros are defined before any header is included, then no symbols other than those defined or reserved for that header by POSIX.1 {3} or the C Standard {2} are made visible. The constraints on the usage of reserved symbols are as specified in POSIX.1 {3}.

See Reason 5 in Section 5 of POSIX.3 {4}.

D01(C) If the implementation supports feature test macros other than {_POSIX_-SOURCE}, and this is documented:
The details are contained in 2.7.2 of the PCD.1.

2.7.2.1 C Standard Language-Dependent Support

There are no assertions specific to this subclause.

2.7.2.2 Common-Usage-Dependent Support

There are no assertions specific to this subclause.

2.7.3 Headers and Function Prototypes

GA36 For all elements with result type not *void* and not *int*:
If the implementation provides C Standard {2} support:
When the header <*.h> is included, then the function prototype *type1 funct*(*type2, type3*) is declared.[5)]

Otherwise:
When the header <*.h> is included, then the function *funct*() is declared with the result type *type1*.

For all elements with result type *void* except *assert*():
If the implementation provides C Standard {2} support:
When the header <*.h> is included, then the function prototype *void funct*(*type2, type3*) is declared.

For all elements with result type *int* except *setjmp*() and *sigsetjmp*():
If the implementation provides C Standard {2} support:
When the header <*.h> is included, then the function prototype *int funct*(*type2, type3*) is declared.

Otherwise:
When the header <*.h> is included, then the function *funct*() is

5) The function prototypes are aligned with ISO/IEC 9945-1: 1990. When not specified in ISO/IEC 9945-1: 1990, they are aligned with ISO/IEC 9899:1990.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 2003.1-1992

either declared with the result type *int* or is not declared in the header.

For *funct*() of *setjmp*() and *sigsetjmp*():

If *funct*() is not defined as a macro:

When the header <setjmp.h> is included, then *funct*() is declared as an identifier with external linkage and result type *int*.

GA37 If *funct*() is defined as a macro when the header <*.h> is included:

When the macro *funct*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), it then expands to an expression with the result type compatible with that specified for *funct*().

There are no assertions specific to this subclause.

2.8 Numerical Limits

2.8.1 C Language Limits

01(A) When the header <limits.h> is included, then the following symbols and corresponding values are defined and have values that meet the requirements of the C Standard {2}, as shown in Table 2-2.

Table 2-2 – C Language Limits

Symbol	Value	Description
CHAR_BIT	8	Minimum number of bits
CHAR_MAX		See note below
CHAR_MIN		See note below
INT_MAX	+32767	Minimum maximum
INT_MIN	-32767	Maximum minimum
LONG_MAX	+2147483647	Minimum maximum
LONG_MIN	-2147483647	Maximum minimum
MB_LEN_MAX	1	Minimum maximum
SCHAR_MAX	+127	Minimum maximum
SCHAR_MIN	-127	Maximum minimum
SHRT_MAX	+32767	Minimum maximum
SHRT_MIN	-32767	Maximum minimum
UCHAR_MAX	255	Minimum maximum
UINT_MAX	65535	Minimum maximum
ULONG_MAX	4294967295	Minimum maximum
USHRT_MAX	65535	Minimum maximum

NOTE: If *char* is signed, then {CHAR_MIN} is equal to {SCHAR_MIN} and {CHAR_MAX} is equal to {SCHAR_MAX}; otherwise, {CHAR_MIN} is zero and {CHAR_MAX} is equal to {UCHAR_MAX}.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 22,2021 at 08:01:05 UTC from IEEE Xplore. Restrictions apply.
Documents Document Sharing Hub
Full document details: IEEE 2003.1-1992

2.8.2 Minimum Values

02(A) When the header <limits.h> is included, then the following symbols and corresponding values are defined as shown in Table 2-3.

Table 2-3 – Minimum Values

Symbol	Value
{_POSIX_ARG_MAX}	4096
{_POSIX_CHILD_MAX}	6
{_POSIX_LINK_MAX}	8
{_POSIX_MAX_CANON}	255
{_POSIX_MAX_INPUT}	255
{_POSIX_NAME_MAX}	14
{_POSIX_NGROUPS_MAX}	0
{_POSIX_OPEN_MAX}	16
{_POSIX_PATH_MAX}	255
{_POSIX_PIPE_BUF}	512
{_POSIX_SSIZE_MAX}	32767
{_POSIX_STREAM_MAX}	8
{_POSIX_TZNAME_MAX}	3

R01 A conforming implementation provides values at least as large as those specified in the previous table for the related symbols of the conforming implementation. (See Assertions 4-15 in 2.8.4 and 2.8.5.)

2.8.3 Run-Time Increasable Values

D01(A) The magnitude limitation fixed for {NGROUPS_MAX} (defined in <limits.h>) is described in 2.8.3 of the PCD.1.

03(A) When the header <limits.h> is included, then the symbolic name {NGROUPS_MAX} is defined, and the value of the symbolic constant is not less than the value of {_POSIX_NGROUPS_MAX}.

2.8.4 Run-Time Invariant Values

04(C) If the symbolic constant {ARG_MAX} is defined:
When the header <limits.h> is included, then the value of {ARG_MAX} is not less than {_POSIX_ARG_MAX}.

05(C) If the symbolic constant {CHILD_MAX} is defined:
When the header <limits.h> is included, then the value of {CHILD_MAX} is not less than {_POSIX_CHILD_MAX}.

06(C) If the symbolic constant {OPEN_MAX} is defined:
When the header <limits.h> is included, then the value of {OPEN_MAX} is not less than {_POSIX_OPEN_MAX}.

07(C) If the symbolic constant {STREAM_MAX} is defined:
When the header <limits.h> is included, then the value of {STREAM_MAX} is not less than {_POSIX_STREAM_MAX}.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

08(C) If the symbolic constant {STREAM_MAX} is defined:
When the header <limits.h> is included, then the value of {STREAM_MAX} is equal to the value of FOPEN_MAX in the header <stdio.h>.

09(C) If the symbolic constant {TZNAME_MAX} is defined:
When the header <limits.h> is included, then the value of {TZNAME_MAX} is not less than {_POSIX_TZNAME_MAX}.

D02(A) The run-time invariant values for {ARG_MAX}, {CHILD_MAX}, {OPEN_MAX}, {STREAM_MAX}, and {TZNAME_MAX} (defined in <limits.h>), are contained in 2.8.4 of the PCD.1.

2.8.5 Pathname Variable Values

10(C) If the symbolic constant {LINK_MAX} is defined:
When the header <limits.h> is included, then the value of {LINK_MAX} is not less than {_POSIX_LINK_MAX}.

11(C) If the symbolic constant {MAX_CANON} is defined:
When the header is included, then the value of {MAX_CANON} is not less than {_POSIX_MAX_CANON}.

12(C) If the symbolic constant {MAX_INPUT} is defined:
When the header <limits.h> is included, then the value of {MAX_INPUT} is not less than {_POSIX_MAX_INPUT}.

13(C) If the symbolic constant {NAME_MAX} is defined:
When the header <limits.h> is included, then the value of {NAME_MAX} is not less than {_POSIX_NAME_MAX}.

14(C) If the symbolic constant {PATH_MAX} is defined:
When the header <limits.h> is included, then the value of {PATH_MAX} is not less than {_POSIX_PATH_MAX}.

15(C) If the symbolic constant {PIPE_BUF} is defined:
When the header <limits.h> is included, then the value of {PIPE_BUF} is not less than {_POSIX_PIPE_BUF}.

D03(A) The values for {LINK_MAX}, {MAX_CANON}, {MAX_INPUT}, {NAME_MAX}, {PATH_MAX}, and {PIPE_BUF} (defined in <limits.h>), conditions under which these values may change, and the limits of such variations, if any, are contained in 2.8.5 of the PCD.1.

2.8.6 Invariant Values

16(D) If the implementation provides a means of obtaining the value of {SSIZE_MAX} at runtime:
The runtime value of {SSIZE_MAX} is equal to the value of SSIZE_MAX defined in <limits.h>.

See Reason 3 in Section 5 of POSIX.3 [4].

17(A) When the header <limits.h> is included, then the macro SSIZE_MAX is defined and evaluates to a value not less than {_POSIX_SSIZE_MAX}.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992

2.9 Symbolic Constants

R01 The implementation has the header <unistd.h>. (See Assertions 1, 3, 5, and 6 in 2.9.1 through 2.9.4.)

D01(A) For constants {_POSIX_JOB_CONTROL} and {_POSIX_SAVED_IDS} (defined in <unistd.h>), the values, conditions under which these values may change, and the limits of such variations, if any, are contained in 2.9 of the PCD.1.

D02(C) If the constant {_POSIX_CHOWN_RESTRICTED} is defined in <unistd.h>:

The value of {_POSIX_CHOWN_RESTRICTED}, conditions under which this value may change, and the limits of such variations, if any, are contained in 2.9 of the PCD.1.

D03(C) If the constant {_POSIX_NO_TRUNC} is defined in <unistd.h>:

The value of {_POSIX_NO_TRUNC}, conditions under which this value may change, and the limits of such variations, if any, are contained in 2.9 of the PCD.1.

D04(C) If the constant {_POSIX_VDISABLE} is defined in <unistd.h>:

The value of {_POSIX_VDISABLE}, conditions under which this value may change, and the limits of such variations, if any, are contained in 2.9 of the PCD.1.

2.9.1 Symbolic Constants for the *access*() Function

01(A) When the header <unistd.h> is included, then the constants R_OK (test for read permission), W_OK (test for write permission), X_OK (test for execute or search permission), and F_OK (test for existence of file) are defined.

02(A) The constants F_OK, R_OK, W_OK, X_OK, and the expressions R_OK | W_OK, R_OK | X_OK, W_OK | X_OK, and R_OK | W_OK | X_OK all have distinct integral values.

2.9.2 Symbolic Constants for the *lseek*() Function

03(A) When the header <unistd.h> is included, then the elements SEEK_SET, SEEK_CUR, and SEEK_END are defined.

04(A) The elements SEEK_SET, SEEK_CUR, and SEEK_END all have distinct integral values.

2.9.3 Compile-Time Symbolic Constants for Portability Specifications

R02 When the header <unistd.h> is included, then the value of the symbol {_POSIX_JOB_CONTROL} is not less restrictive than that provided by the respective POSIX option via *sysconf*(). (See Assertion 12 in 4.8.1.2.)

R03 When the header <unistd.h> is included, then the value of the symbol {_POSIX_SAVED_IDS} is not less restrictive than that provided by the respective POSIX option via *sysconf*(). (See Assertion 13 in 4.8.1.2.)

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25,2026 at 03:43:07 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: IEEE 2003.1-1992

05(A) When the header <unistd.h> is included, then {_POSIX_VERSION} is defined and matches the integer value of POSIX.1 {3}, "199009L."

2.9.4 Execution-Time Symbolic Constants for Portability Specifications

D05(C) If {_POSIX_CHOWN_RESTRICTED} is not defined or defined with a value other than -1 in <unistd.h>:

The files and file types to which {_POSIX_CHOWN_RESTRICTED} applies are contained in 2.9.4 of the PCD.1.

R04 When the header <unistd.h> is included, and the value of the symbol {_POSIX_CHOWN_RESTRICTED} is defined and has a value other than -1, then the value of the symbol provides the same indication as that provided by the interrogation at run-time of the POSIX.1 {3} option via *pathconf*() and *fpathconf*(). (See Assertions 36 and 20 in 5.7.1.1.2 and 5.7.1.2.2.)

D06(C) If {_POSIX_NO_TRUNC} is not defined or defined with a value other than -1 in <unistd.h>:

The files and file types to which {_POSIX_NO_TRUNC} applies are contained in 2.9.4 of the PCD.1.

R05 When the header <unistd.h> is included, and the value of the symbol {_POSIX_NO_TRUNC} is defined and has a value other than -1, then the value of the symbol provides the same indication as that provided by the interrogation at run-time of the POSIX.1 {3} option via *pathconf*() and *fpathconf*(). (See Assertions 38 and 22 in 5.7.1.1.2 and 5.7.1.2.2.)

D07(C) If {_POSIX_VDISABLE} is not defined or defined with value other than -1 in <unistd.h>:

The files and file types to which {_POSIX_VDISABLE} applies are contained in 2.9.4 of the PCD.1.

R06 When the header <unistd.h> is included, and the value of the symbol {_POSIX_VDISABLE} is defined and has a value other than -1, then the value of the symbol provides the same indication as that provided by the interrogation at run-time of the POSIX.1 {3} option via *pathconf*() and *fpathconf*(). (See Assertions 40 and 24 in 5.7.1.1.2 and 5.7.1.2.2.)

06(D) If {_POSIX_CHOWN_RESTRICTED} is defined with the value -1 when <unistd.h> is included:

When the header <unistd.h> is included and {_POSIX_CHOWN_RESTRICTED} is -1, then the implementation does not provide this option.

See Reason 3 in Section 5 of POSIX.3 {4}.

07(D) If {_POSIX_NO_TRUNC} is defined with the value -1 when <unistd.h> is included:

When the header <unistd.h> is included and {_POSIX_NO_TRUNC} is -1, then the implementation does not provide this option.

See Reason 3 in Section 5 of POSIX.3 {4}.

08(D) If {_POSIX_VDISABLE} is defined with the value -1 when <unistd.h> is included:

Authorized licensed use limited to: Tsinghua University. Downloaded on February 22,2022 at 08:01:01 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 2003.1-1992
AVA Documents Document Sharing Hub