

Abstract: This standard is part of the POSIX series of standards for applications and user interfaces to open systems. It defines the Ada language bindings as package specifications and accompanying textual descriptions of the application program interface (API). This standard supports application portability at the source code level through the binding between ISO 8652: 1987 (Ada) and ISO/IEC 9945-1: 1990 (IEEE Std 1003.1-1990) (POSIX), as amended by IEEE Std 1003.1b-1993, IEEE Std 1003.1c-1995, and IEEE Std 1003.1i-1995. Terminology and general requirements, process primitives, the process environment, files and directories, input and output primitives, device- and class-specific functions, language-specific services for Ada, system databases, synchronization, memory management, execution scheduling, clocks and timers, and message passing are covered. It also specifies behavior to support the binding that must be provided by the Ada compilation system.

Keywords: Ada, API, application portability, computer language bindings, open systems, operating systems, portable application, POSIX, POSIX language bindings, real-time, thread.

POSIX is a trademark of the Institute of Electrical and Electronics Engineers, Inc

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

Copyright © 1999 by the
Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1996.
Printed in the United States of America.

ISBN 0-7381-1570-3

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.

18 March 1999

SH94726

IEEE Standards documents are developed within 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 societies and Standards Coordinating 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

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying all patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (508) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Contents

	PAGE
Introduction	xi
Section 1: General	1
1.1 Scope	1
1.2 Normative References	2
1.3 Conformance	3
1.3.1 Implementation Conformance	3
1.3.2 Application Conformance	6
Section 2: Terminology and General Requirements	7
2.1 Editorial Conventions	7
2.2 Definitions	8
2.2.1 Terminology	8
2.2.2 General Terms	10
2.2.3 Abbreviations	26
2.3 General Concepts	27
2.3.1 Process/Active Partition Relationship	27
2.3.2 Task/Thread Relationship	27
2.3.3 Ada Character Differences	27
2.3.4 Posix Signals Are Not Interrupts	28
2.3.5 System Call Exception Errors	28
2.3.6 Extended Security Controls	28
2.3.7 File Access Permissions	28
2.3.8 File Hierarchy	29
2.3.9 Filename Portability	29
2.3.10 File Times Update	29
2.3.11 Pathname Resolution	30
2.3.12 Allocated Storage	31
2.4 Package POSIX	31
2.4.1 Constants and Static Subtypes	36
2.4.2 POSIX Characters	41
2.4.3 POSIX Strings	42
2.4.4 String Lists	43
2.4.5 Option Sets	45
2.4.6 Error Codes and Exceptions	46
2.4.7 System Identification	52
2.4.8 Time Types	53
2.5 Package POSIX_Options	55
2.5.1 Implementation Options	55

2.6	Package POSIX_Limits	58
2.6.1	Implementation Limits	60
2.7	Package Ada_Streams	64
2.8	Package System	65
2.9	Package System_Storage_Elements	66
2.10	Package POSIX_Page_Alignment	67
2.10.1	Description	67
2.10.2	Error Handling	68
2.11	Environment Description	68
2.11.1	Time-Zone Information	69
Section 3: Process Primitives		71
3.1	Package POSIX_Process_Primitives	71
3.1.1	Process Template	73
3.1.2	Process Creation	75
3.1.3	Process Exit	79
3.1.4	Termination Status	80
3.1.5	Wait for Process Termination	82
3.2	Package POSIX_Unsafe_Process_Primitives	84
3.2.1	Process Creation	85
3.2.2	File Execution	87
3.3	Package POSIX_Signals	91
3.3.1	Signal Model	94
3.3.2	Signal Type	98
3.3.3	Standard Signals	98
3.3.4	Default Signal Actions	102
3.3.5	Tasking Safety	103
3.3.6	Interruptibility	103
3.3.7	Signal Sets	105
3.3.8	Block and Unblock Signals	106
3.3.9	Ignore Signals	107
3.3.10	Controlling Generation of Signal for Child Process	108
3.3.11	Examine Pending Signals	109
3.3.12	Signal Event Notification	109
3.3.13	Signal Information	110
3.3.14	Control Signal Queueing	112
3.3.15	Wait for Signal	113
3.3.16	Wait for Signal with Information	115
3.3.17	Signal Entries	116
3.3.18	Send a Signal	118
3.3.19	Queue a Signal	119
3.3.20	Interrupt a Task	120
Section 4: Process Environment		123
4.1	Package POSIX_Process_Identification	123
4.1.1	Process Identification Operations	124
4.1.2	Process Group Identification	125
4.1.3	User Identification	127

4.1.4	User and Group Identification	128
4.2	Package <code>POSIX_Process_Times</code>	130
4.2.1	Process Time Accounting	130
4.3	Package <code>POSIX_Process_Environment</code>	132
4.3.1	Argument List	133
4.3.2	Environment Variables	133
4.3.3	Process Working Directory	137
4.4	Package <code>POSIX_Calendar</code>	138
4.4.1	Time Information	139
4.4.2	Operations on POSIX Times	140
4.5	Package <code>POSIX_Configurable_System_Limits</code>	142
4.5.1	Get Configurable System Options	143
4.5.2	Get Configurable System Limits	145
Section 5: Files and Directories		149
5.1	Package <code>POSIX_Permissions</code>	149
5.1.1	File Permissions	150
5.1.2	Process Permission Set	151
5.2	Package <code>POSIX_Files</code>	152
5.2.1	Create and Remove Files	153
5.2.2	Inquiries on File Types	156
5.2.3	Modify File Pathnames	156
5.2.4	Directory Iteration	159
5.2.5	Update File Status Information	161
5.2.6	Check File Accessibility	163
5.3	Package <code>POSIX_File_Status</code>	164
5.3.1	Access File Status	165
5.3.2	Access Status Information	166
5.4	Package <code>POSIX_Configurable_File_Limits</code>	168
5.4.1	File Limits	170
5.4.2	File Restrictions	174
5.4.3	Pathname-Specific Options	176
Section 6: Input and Output Primitives		179
6.1	Package <code>POSIX_IO</code>	179
6.1.1	Open or Create a File	181
6.1.2	I/O Buffer Type	187
6.1.3	Read from a File	187
6.1.4	Write to a File	190
6.1.5	File Position Operations	195
6.1.6	Terminal Operations	196
6.1.7	File Control	196
6.1.8	Update File Status Information	198
6.1.9	Truncate File to A Specified Length	199
6.1.10	Synchronize a File's State	200
6.1.11	Data Synchronization	201
6.2	Package <code>POSIX_File_Locking</code>	202
6.2.1	Lock and Unlock a Region of a File	203

6.3	Package <code>POSIX_Asyncronous_IO</code>	204
6.3.1	AIO Descriptor Type	206
6.3.2	Attributes of AIO Control Blocks	208
6.3.3	Asynchronous Read	211
6.3.4	Asynchronous Write	212
6.3.5	List Directed I/O	214
6.3.6	Retrieve Status of Asynchronous I/O Request	216
6.3.7	Retrieve Bytes Transferred by Asynchronous I/O Request	218
6.3.8	Cancel Asynchronous I/O Request	219
6.3.9	Wait for Asynchronous I/O Request to Complete	220
6.3.10	Asynchronous File and Data Synchronization	221
Section 7: Device- and Class-Specific Functions		225
7.1	General Terminal Interface	225
7.1.1	Interface Characteristics	225
7.2	Package <code>POSIX_Terminal_Functions</code>	232
7.2.1	Terminal Characteristics	234
7.2.2	Input Modes	237
7.2.3	Output Modes	239
7.2.4	Control Modes	239
7.2.5	Local Modes	240
7.2.6	Retrieve and Define Terminal Modes and Bits per Character	242
7.2.7	Special Control Characters	243
7.2.8	Noncanonical Controls	245
7.2.9	Baud Rate Subprograms	245
7.2.10	Line Control Operations	246
7.2.11	Foreground Process Group ID	248
7.2.12	Generate Terminal Pathname	249
Section 8: Language-Specific Services for Ada		251
8.1	Interoperable Ada I/O Services	251
8.1.1	Requirements	251
8.1.2	Additional Interpretation of the POSIX.1 Standard	255
8.2	Package <code>POSIX_Supplement_to_Ada_IO</code>	256
8.2.1	Parse Form values	257
8.2.2	Flush Files	258
Section 9: System Databases		259
9.1	Package <code>POSIX_User_Database</code>	259
9.1.1	Access Contents of a User Database Item	259
9.1.2	Access User Database Items	260
9.2	Package <code>POSIX_Group_Database</code>	261
9.2.1	Access Contents of a Group Database Item	262
9.2.2	Access Elements of the Group Item List of Members	262
9.2.3	Access Group Database Items	263
Section 10: Data Interchange Format		265
Section 11: Synchronization		267

11.1	Package <code>POSIX_Semaphores</code>	267
11.1.1	Semaphore and Semaphore Descriptor Types	268
11.1.2	Initialize an Unnamed Semaphore	269
11.1.3	Finalize an Unnamed Semaphore	270
11.1.4	Create/Open a Named Semaphore	271
11.1.5	Close a Named Semaphore	273
11.1.6	Remove a Named Semaphore	274
11.1.7	Decrement a Semaphore	275
11.1.8	Increment a Semaphore	276
11.1.9	Get the Value of a Semaphore	276
11.2	Package <code>POSIX_Mutexes</code>	277
11.2.1	Mutex Ownership	278
11.2.2	Mutex and Mutex Descriptor Types	279
11.2.3	Mutex Attributes Type	279
11.2.4	Mutex Process Shared Attribute	280
11.2.5	Mutex Locking Policy Attributes	281
11.2.6	Initialize and Finalize a Mutex	283
11.2.7	Change the Ceiling Priority of a Mutex	285
11.2.8	Lock and Unlock a Mutex	286
11.3	Package <code>POSIX_Condition_Variables</code>	287
11.3.1	Condition and Condition Descriptor Types	288
11.3.2	Condition Variable Attributes Type	288
11.3.3	Condition Process Shared Attribute	289
11.3.4	Initialize and Finalize a Condition	290
11.3.5	Broadcast and Signal a Condition	292
11.3.6	Wait on a Condition	293
	Section 12: Memory Management	295
12.1	Package <code>POSIX_Memory_Locking</code>	296
12.1.1	Lock/Unlock a Process's Address Space	296
12.2	Package <code>POSIX_Memory_Range_Locking</code>	298
12.2.1	Lock/Unlock a Range of Process Address Space	298
12.3	Package <code>POSIX_Memory_Mapping</code>	300
12.3.1	Map Process Addresses to a Memory Object	301
12.3.2	Unmap Memory	305
12.3.3	Change Memory Protection	306
12.3.4	Memory Object Synchronization	308
12.4	Package <code>POSIX_Shared_Memory_Objects</code>	309
12.4.1	Open a Shared Memory Object	310
12.4.2	Remove a Shared Memory Object	312
12.5	Package <code>POSIX_Generic_Shared_Memory</code>	313
12.5.1	Open Shared Memory	314
12.5.2	Access Shared Memory	317
12.5.3	Close Shared Memory	317
12.5.4	Remove Shared Memory	318
12.5.5	Lock/Unlock Shared Memory	318
	Section 13: Execution Scheduling	321

13.1	Scheduling Concepts and Terminology	321
13.2	Package <code>POSIX_Process_Scheduling</code>	321
13.2.1	Scheduling Parameters	322
13.2.2	Scheduling Policies	323
13.2.3	Modify Process Scheduling Policy and Parameters	323
13.2.4	Process Yield CPU	325
13.2.5	Get Scheduling Limits	325
13.3	Task Scheduling	326
13.3.1	Dynamic Priorities	326
13.3.2	Task Dispatching Policy Pragma	326
13.3.3	Task Creation Attributes Pragma	327
13.3.4	Task Yield CPU	327
13.4	Synchronization Scheduling	328
Section 14: Clocks and Timers		329
14.1	Package <code>POSIX_Timers</code>	329
14.1.1	Clock and Timer Types	330
14.1.2	Realtime Clock	330
14.1.3	Timer State and Timer Options	331
14.1.4	Clock Operations	332
14.1.5	Create a Timer	333
14.1.6	Delete a Timer	335
14.1.7	Timer Operations	335
14.2	High Resolution Delay	338
Section 15: Message Passing		339
15.1	Package <code>POSIX_Message_Queues</code>	339
15.1.1	Message Queue Attributes	341
15.1.2	Open a Message Queue	343
15.1.3	Close a Message Queue	346
15.1.4	Remove a Message Queue	347
15.1.5	Send a Message to a Message Queue	348
15.1.6	Receive a Message from a Message Queue	349
15.1.7	Generic Message Passing	350
15.1.8	Notify Process that a Message is Available	352
15.1.9	Set/Query Message Queue Attributes	353
Section 16: Task Management		355
16.1	Package <code>Ada_Task_Identification</code>	355
16.1.1	Description	355
Annex A (informative) Bibliography		357
Annex B (informative) Rationale and Notes		359
B.1	General	359
B.1.1	Purpose and Audience	359
B.1.2	Relation to Other POSIX Standards	359
B.1.3	Ada Language	360
B.1.4	Implementation Model	361

B.1.5	Level of Binding	361
B.1.6	Form of Document	362
B.1.7	Global Issues	363
B.1.8	Naming	364
B.1.9	Mapping C Features to Ada	365
B.1.10	Conformance	370
B.2	Terminology and General Requirements	370
B.2.1	Conventions	370
B.2.2	Definitions	370
B.2.3	General Concepts	371
B.2.4	Package POSIX	374
B.2.5	Package POSIX_Options	380
B.2.6	Package POSIX_Limits	381
B.2.7	Package Ada_Streams	381
B.2.8	Package System	381
B.2.9	Package System_Storage_Elements	381
B.2.10	Package POSIX_Page_Alignment	381
B.2.11	Environment Description	382
B.3	Process Primitives	382
B.3.1	Package POSIX_Process_Primitives	382
B.3.2	Process Creation	383
B.3.3	Process Exit	384
B.3.4	Wait for Process Termination	386
B.3.5	Package POSIX_Unsafe_Process_Primitives	386
B.3.6	Package POSIX_Signals	386
B.3.7	Signal Model	387
B.3.8	Signal Masking and Related Concepts	387
B.3.9	Tasking Safety	391
B.3.10	Signal Type	392
B.3.11	Standard Signals	392
B.3.12	Signal Sets	393
B.3.13	Examine and Change Signal Action	393
B.3.14	Signal Queueing	394
B.3.15	Signal Notification Model	395
B.3.16	Examine Pending Signals	396
B.3.17	Send a Signal	396
B.3.18	Binding for <i>sigsuspend</i>	396
B.3.19	Synchronously Accept a Signal	396
B.3.20	Queue a Signal to a Process	397
B.3.21	Send a Signal to a Thread	397
B.3.22	Schedule Alarm	398
B.3.23	Suspend Process Execution	398
B.3.24	Delay Process Execution	398
B.3.25	Task Signal Entries	398
B.3.26	Composability Considerations	399
B.4	Process Environment	400
B.4.1	Organization	400
B.4.2	Package POSIX_Process_Identification	400

B.4.3	Package Process_Times	401
B.4.4	Package POSIX_Process_Environment	402
B.4.5	Package POSIX_Calendar	404
B.4.6	Package POSIX_Configurable_System_Limits	405
B.5	Files and Directories	405
B.5.1	Organization	405
B.5.2	Package POSIX_Permissions	406
B.5.3	Package POSIX_Files	407
B.5.4	Package POSIX_File_Status	408
B.5.5	Package POSIX_Configurable_File_Limits	408
B.5.6	Directory Operations	409
B.6	Input and Output Primitives	409
B.6.1	Package POSIX_IO	409
B.6.2	Package POSIX_File_Locking	414
B.6.3	Package POSIX_Asynchronous_IO	415
B.7	Device- and Class-Specific Functions	417
B.8	Language-Specific Services for Ada	418
B.8.1	General Rationale	418
B.8.2	Limitations on Interoperability	420
B.8.3	Rationale for Form Parameter	421
B.8.4	Unaffected Implementation Dependencies	426
B.8.5	Notes on Specific Topics	426
B.9	System Databases	428
B.9.1	Requirements from POSIX.1	428
B.9.2	Rationale for the Current Design	429
B.10	Data Interchange Format	430
B.11	Synchronization	430
B.11.1	Package POSIX_Semaphores	430
B.11.2	Mutexes & Condition Variables	433
B.12	Memory Management	436
B.12.1	Process Memory Locking	436
B.12.2	Package POSIX_Memory_Mapping	437
B.12.3	Package POSIX_Shared_Memory_Objects	438
B.12.4	Package POSIX_Generic_Shared_Memory	439
B.13	Priority Process Scheduling	441
B.13.1	Scheduling Concepts and Terminology	442
B.13.2	Package POSIX_Process_Scheduling	442
B.13.3	Task Scheduling	443
B.13.4	Synchronization Scheduling	444
B.13.5	Thread Scheduling Pragmas vs. Environment Variables	445
B.14	Clocks and Timers	446
B.14.1	Types and Constants	446
B.14.2	Clock Operations	447
B.14.3	Timer Creation	447
B.14.4	Timer Operations	447
B.14.5	High Resolution Sleep	448
B.15	Message Passing	448
B.15.1	Message Queue Attributes	449

B.15.2	Priorities	449
B.15.3	Generic Message Passing	449
B.15.4	Notification	450
B.16	Task Identification	450
B.17	Thread-Specific Data	450
Annex C (informative)	Ada/C Cross-References	451
C.1	Ada-to-C Cross-Reference	451
C.2	C-to-Ada Cross-Reference	469
Alphabetic Topical Index		489

TABLES

Table2.1	– Typographical Conventions	7
Table2.2	– Constant and Subtype Correspondences	37
Table2.3	– Option Set Comparisons	46
Table2.4	– Static Subtypes and Options	59
Table2.5	– Portable Constants and Limits	63
Table2.6	– Static Subtypes and Limits	64
Table3.1	– Default Actions for Job Control Signals	103
Table4.1	– Functions for System-Wide Options	145
Table4.2	– Configurable System Limits	146
Table6.1	– Standard File Descriptors	182
Table6.2	– Error Codes and AIO Status Values	217
Table7.1	– Terminal_Characteristics Components	235
Table7.2	– Terminal_Modes Values for Input Control	237
Table7.3	– Terminal_Modes Values for Output Control	239
Table7.4	– Terminal_Modes Values for Hardware Control	240
Table7.5	– Terminal_Modes Values for Local Control Modes	241
Table7.6	– Special Control Character Usage	244

International Standard ISO/IEC 14519: 1999(E)

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

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

International Standard ISO/IEC 14519 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC22, *Programming languages, their environments and system software interfaces*.

Annexes A through C of ISO/IEC14519 are for information only.

Introduction

(This introduction is not a part of IEEE Std 1003.5b-1996, IEEE Standard for Information Technology—POSIX Ada Language Interfaces—Part 1: Binding for System Application Program Interface (API) Amendment 1: Realtime Extensions but is included for information only.)

This standard is an amended version of IEEE Std 1003.5-1992. The basic goal of this standard is to provide an Ada application program interface for the language-independent services made accessible to C-language applications programs by the interfaces defined in ISO/IEC 9945-1: 1990 (IEEE Std 1003.1-1990), as amended by IEEE Std 1003.1b-1993, IEEE Std 1003.1c-1995, and IEEE Std 1003.1i-1995.

The intent is to support portability of Ada applications via a standard binding to the services provided by a POSIX-conforming operating system. POSIX is defined by the standard C-language interfaces cited above. Therefore, much of the work in producing this standard was deciding what features of those C-language interfaces represented POSIX functionality, as opposed to C-language specific features.

This standard provides package specifications and accompanying textual description for a set of Ada packages that represent the POSIX system. This standard also specifies behavior to support the binding that must be provided by the Ada compilation system, and further defines behavior specified as implementation defined in the Ada Language standard (particularly in the area of Text_IO) for use in a POSIX environment.

The emphasis in POSIX is on application program portability, so the interfaces in this standard are not sufficient to implement an Ada compilation system or a POSIX shell (as defined in IEEE Std 1003.2-1992). For an application, the intent is that a Strictly Conforming POSIX.5 Application (one that uses only the facilities in this standard and that does not depend on implementation defined behavior) can be ported to any Conforming Implementation of these interfaces, and that the binding makes it easy to identify where a program is not strictly conforming and makes such programs easier to port.

Organization of This Standard

The standard is divided into three parts:

- Statement of scope, list of normative references, and conformance information (Section 1)
- Definitions and global concepts (Section 2)
- The various interface facilities (Sections 3 through 16)

The content of the sections parallels that of the correspondingly numbered sections of the base standards (IEEE Std 1003.1-1990, as amended by IEEE Std 1003.1b-1993,

IEEE Std 1003.1c-1995, and IEEE Std 1003.1i-1995), with some changes required to fit the style of IEEE Std 1003.5-1993. There is no Section 10, since there is no Ada binding for that section (Data Interchange Formats) of the base standards.

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

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 descriptions of the services defined by IEEE-Std 1003.1-1990 and its amendments.
- (2) C and FORTRAN language bindings to (1)
- (3) Shell and utility facilities
- (4) Verification testing methods
- (5) Secure/Trusted system considerations
- (6) Network interface facilities
- (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 standard, following IEEE and ISO/IEC procedures.

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

If you have an interest in participating in the PASC working groups addressing these issues, please send your name, address, and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, USA, and ask to have this forwarded to the chair of the appropriate PASC working group. If you have an interest in participating in this work at the international level, contact your ISO/IEC national body.

¹A *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, USA; Telephone: +1 202 371-0101; FAX: +1 202 728-9614.

IEEE Std 1003.5-1992 was prepared by the 1003.5 Working Group, sponsored by the Technical Committee on Operating Systems and Applications Environments of the IEEE Computer Society. At the time this standard was approved, the membership of the IEEE P1003.5 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 Kevra

Pete Meier

Andrew Twigger

Treasurer: Peter Smith

Secretary: Shane McCarron

P1003.5 Working Group Officials

Chair: James P. Lonjers

Steven Deller (1989-1991)

Major Terrence Fong (1988-1989)

Vice Chairs: James P. Lonjers (1990-1991)

Major Terrence Fong (1989-1990)

Stowe Boyd (1988-1989)

Editors: David Emery

Hal Jespersen

Steven Deller (1988-1989)

Rationale Editor: Mitch Gart

Secretary: C. Jayne Baker

David Emery (1988-1989)

Technical Reviewers

Ted Baker

David Emery

Jim Lonjers

Steven Deller

Mitch Gart

Jim Moore

Dennis Doubleday

Stephen Schwarm

Working Group

Ted Baker

Michael Gillam

Sue LeGrand

Stowe Boyd

Al Globus

James Lonjers

Bevin Brett

Mars Gralia

James Moore

Charles Brown

Jayne Guyse

Mark Ruddock

Robert Brown

Ken Harvey

Stephen Schwarm

Bhaves Damania

Ruth Hirt

Michael Shapiro

Steven Deller

Jeff Hooley

Brian Sullivan

Dennis Doubleday

Michael Kjolsrud

Del Swanson

David Emery

Peter Krupp

Robert Voigt

Terry Fong

James Leathrum

Olle Wikstrom

Mitchell Gart

John Zenor

1 The following persons were members of the 1003.5 balloting group that approved the
 2 standard for submission to the IEEE Standards Board:

3	Harold C. Adams	Andrew Chung	Allen L. Grau
4	John S. Adams	Brad Clark	Charles R. Grauling
5	Omar Ahmed	Lori A. Clarke	Daniel Green
6	David Allen	Norman H. Cohen	Tom Griest
7	Charles J. Antonelli	Edward Colbert	F. Grize
8	B. Ardary	Phillippe Collard	Ernesto Guerrieri
9	David Athersych	Robert A. Conti	Lawrence M. Gunther
10	Randall Atkinson	William M. Corwin	R. N. Hagen
11	Randal J. August	Mike Cossey	Charles Hammons
12	Kenneth A. Austin	John Courtney	Peter A. Hansen
13	Carolyn J. Baker	Donald Cragun	Sam Harbaugh
14	Robert L. Baker	Richard A. Crawford	Samuel Harbison
15	Ted Baker	Jim Creegan	David S. Hardin
16	James Baldo	Phyllis Crill	Charles Harkey
17	Brad Balfour	John J. Cupak	Loren L. Hart
18	Gary E. Barnes	Charles Dana	Thomas S. Hawker
19	Mitchell C. Barnhart	William H. Dashiell	Clark M. Hay
20	Randall Barron	David Davis	Ralph Hayward
21	Steven Barryte	Rich DeBernardo	John Craig Heberle
22	Barbara K. Beauchamp	David DeFanti	William Hefley
23	Gary Beerman	Mike Dean	A. Marlow Henne
24	E. Jerome Bell	Dave Decot	Donald C. Hill
25	Donald Bennett	Steven Deller	Norman Hines
26	Peter A. Berggren	Jorge Diaz-Herrera	C. Michael Holloway
27	Mark Biggar	Michael B. Dillencourt	Jeffrey Hooley
28	Robert Bismuth	James H. Dobbins	Joseph P. Hoolihan
29	Alex Blakemore	Audrey Dorofee	Tom Housman
30	Stephen Blanchette, Jr.	Terence Dowling	Richard Howard
31	Pieter Botman	Diptendu Dutta	Norman R. Howes
32	Stowe Boyd	Eugene Edelstein	Lynne M. Hubbs
33	Carl Brandon	Theodore F. Elbert	David K. Hughes
34	Philip Brashear	Richard W. Elwood	Richard G. Hull
35	Joseph P. Brazy	David Emery	Jeremy James
36	Mark S. Breckenridge	Arny Engelson	Hal Jespersen
37	Ronald F. Brender	Philip H. Enslow	Darryl N. Johnson
38	Jim Briggs	William Eventoff	Bruce Johnston
39	Thomas C. Brooke	Gary Falacara	Alain Jouchoux
40	Jerry R. Brookshire	John H. Fauerby	Juern Juergens
41	Charles O. Brown	Charles A. Finnell	Steven Kahn
42	Elizabeth B. Brown	Jeffery Fischer	Fumimiko Kamijo
43	Jane C. Bryan	Shayne Flint	Alan Kaminsky
44	Gary L. Burt	Terence Fong	Ling Kan
45	Christopher Byrnes	Edward J. Forbes, Jr.	Karl Kelley
46	David Calloway	Roy S. Freedman	Robert H. C. Kemp
47	Nicholas A. Camillone	Randal S. Freier	Judy S. Kerner
48	Kenneth W. Campbell	Dale J. Gaumer	James J. Keys
49	Rick Carle	Larry Gearhart	Paul J. King
50	David J. Carlson	K. M. George	Hans R. Klay
51	Dana Carson	Gregory A. Gicca	Kenneth Kloss
52	Jeffrey R. Carter	Robert T. Goettge	Robert Knighten
53	Jerry Cashin	Phillip Goldstein	Joseph B. Kolb
54	H. L. Catala	Roger Golliver	John C. Krasnowski
55	Larry Chandler	William N. Goolsby	Lak Ming Lam
56	Andy Cheese	William J. Goulet	Rudolf C. Landwehr
57	James Chelini	Mars J. Gralia	Charles F. Lanman

1	Gary Lauther	James K. Parrish	Ronald Skoog
2	Patricia K. Lawlis	Thomas Parrish	Thomas J. Smith
3	Scott A. Leschke	Offer Pazy	Charles Snyder
4	M. Levitz	Walt Penney	Jon S. Squire
5	Stephen H. Levy	Guido Persch	Jeff Stevenson
6	F. C. Lim	Flavio Petersen	Brian Sullivan
7	Timothy E. Lindquist	Thomas A. Peterson	Del Swanson
8	J. J. Logan	George W. Petznick	S. Tucker Taft
9	James P. Lonjers	Hane W. Polzer	Ravi Tavakley
10	Warren E. Loper	J. Pottmyer	Donn S. Terry
11	Mark Loveland	Charles Pow	John A. Thalhamer
12	George A. Ludgate	Eileen Quann	William J. Thomas
13	Sonny Lundahl	Paul Rabin	Peter L. Thompson
14	Wesley Mackey	John Reddan	James L. Troy
15	Austin J. Maher	W. Scott Redmon	Roger Tubby
16	James Maloney	Gregg Reed	Mark-Rene Uchida
17	Roger Martin	Carl Reinert	L. David Umbaugh
18	Robert Mathis	Judith Richardson	Robert B. Urling
19	Fred Maymir-Ducharme	Richard A. Rink	Evelyn M. Uzzle
20	Catherine McDonald	Clyde Roby	Frances Van Scoy
21	Robert L. McGarvey	C. Allan Rofer	Leonard Vanek
22	Daniel L. McNamee	Hyman Rosen	Michael W. Vannier
23	Robert McWhirter	Jerome D. Rosen	Uwe Wacker
24	Nancy R. Mead	Frederick M. Rysz	Robert N. Wagoner
25	Geoff Mendal	Agnes M. Sardi	Mary Wall
26	Jay Michael	Robert J. Satnik	Stephen R. Walli
27	Gary W. Miller	Allen Saxton	Neal Walters
28	Robert E. Miller	Lorne H. Schachter	Kenneth Wasmundt
29	Judah Mogilensky	F. P. Schauer	William Webster
30	Al Mok	Alfred H. Scholldorf	J. Richard Weger
31	Charles S. Mooney	Ron Schroeder	Brian Weis
32	James D. Mooney	Mike Schultz	Robert Weissensee
33	Freeman Moore	W. L. Schultz	Michael K. Welter
34	James W. Moore	Fritz Schulz	Stephen Wersan
35	Jerry A. Moore	Leonard Seagren	Thomas Wheeler
36	John I. Moore, Jr.	Richard Seibel	William Whitaker
37	Duncan Morrill	Lawrence H. Shafer	Bruce Wieand
38	M. W. Morron	Michael D. Shapiro	David Willcox
39	Gary Mrenak	John G. Shea	David C. Willet
40	David G. Mullens	Nagy M. Shehad	David Williamson
41	Richard E. Nese	Dan Shia	Paul A. Willis
42	Sai Lun Ng	Thomas E. Shields	David H. Winfield
43	Daniel Nissen	Keith Shillington	David C. Wolfe
44	Karl Nyberg	David Shochat	Paul A. Wolfgang
45	James O'Day	Stephen Schwarm	Michal Young
46	Evelyn Obaid	Robert Charles Shock	Oren Yuan
47	Patricia Oberndorf	Jerome L. Sibol, Jr.	Janusz Zalewski
48	Kurt M. Olender	Lee Silverthorn	K. M. Zemrowski
49	S. Ron Oliver		John Zolnowsky

When the IEEE Standards Board approved this standard on 20 June 1996, it had the following membership:

Donald C. Loughry, *Chair*

Richard J. Holleman, *Vice Chair*

Andrew G. Salem, *Secretary*

Gilles A. Baril	E. G. “Al” Kiener	Jose R. Ramos
Clyde R. Camp	Joseph L. Koepfinger*	Arthur K. Reilly
Joseph A. Cannatelli	Stephen R. Lambert	Ronald H. Reimer
Stephen L. Diamond	Lawrence V. McCall	Gary S. Robinson
Harold E. Epstein	Bruce McClung	Ingo Rüsçh
Donald C. Fleckenstein	Marco W. Migliaro	John S. Ryan
Jay Forster *	Mary Lou Padgett	Chee Kiow Tan
Donald N. Heirman	John W. Pope	Leonard L. Tripp
Ben C. Johnson		Howard L. Wolfman

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
 Alan H. Cookson
 Chester C. Taylor
 Lisa S. Young
IEEE Standards Project Editor

The POSIX.5b standard was prepared by the P1003.5 working group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society. At the time this standard was approved the membership of the P1003.5 working group was as follows:

Portable Applications Standards Committee (PASC)

Chair: Jehan-François Pâris

PASC Standards Subcommittee

Chair: Lowell Johnson
 Vice Chair: Charles Severance
 Functional Chairs: Jay Ashford
 Andrew Josey
 Barry Needham
 Stephen R. Walli
 Secretary: Nick Stoughton

IEEE P1003.5 Working Group Officials

Chair: James P. Lonjers (1991-1994)
 Stephen Schwarm (1995-1996)
 Ted Baker (1996-1997)
 Vice Chairs: Stephen Schwarm (1991-1995)
 Randy Greene (1995-1996)
 David Emery (1003.5 Interpretations)
 Editor: Ted Baker (P1003.5b)
 Rationale Editor: Lee Lucas
 Secretary: C. Jayne Guyse (1991-1993)
 Peter Obermayer (1994-1996)

Technical Reviewers

Ted Baker	Peter Obermayer	Henry H. Robbins
Mark Faulk	Offer Pazy	Stephen Schwarm
Ted Giering	Ruth A. Peek	Del Swanson
Randy Greene	Ed Posnak	Laurent Visconti
Lee Lucas		

Working Group

Theodore P. (Ted) Baker	David K. Hughes	Ray Ricco
Bevin Brett	James Lonjers	Henry H. Robbins
Steven Deller	Lee Lucas	Stephen Schwarm
David Emery	Peter Obermayer	Jim Smith
Mark Faulk	James T. Oblinger	Del Swanson
Randy Greene	Offer Pazy	Laurent Visconti
C. Jayne Guyse	Ruth A. Peek	John Zenor

1 The following persons were members of the 1003.5b balloting group that approved
2 the standard for submission to the IEEE Standards Board:

3	Alejandro A. Alonso	Norman R. Howes	Dave Plauger
4	Theodore P. Baker	David K. Hughes	Arlan Pool
5	Robert Barned	Judy Kerner	Henry H. Robbins
6	Andy Bihain	Philippe Kruchten	Stephen Schwarm
7	William M. Corwin	Thomas M. Kurihara	Leonard W. Seagren
8	Steven Deller	Arthur Licht	Robert Alan Siegel
9	David Emery	C. Douglass Locke	Dennis C. Stewart
10	Philip H. Enslow	James P. Lonjers	Alfred Strohmeier
11	Michael Gonzalez	Lee W. Lucas	Del Swanson
12	C. Jayne Guyse	Roland McGrath	Mark-René Uchida
13	Joe Gwinn	Paul Murdock	USENIX
14	Patrick Hebert	James T. Oblinger	Victor Fay-Wolfe
15	Steven Howell	Offer Pazy	John Zenor

Information technology— POSIX[®] Ada Language Interfaces— Part 1: Binding for System Application Program Interface (API)— Amendment 1: Realtime Extensions

Section 1: General

1.1 Scope

This standard defines a standard Ada language binding to POSIX.1 {2}, as amended by POSIX.1b {3}, POSIX.1c {4}, and POSIX.1i, which are hereinafter called *the base standards*. (See §1.2.) The base standards define C-language interfaces to operating system services.¹ This standard defines interfaces by which the same operating system services may be accessed from the Ada programming language.

The purpose of this standard is to support application portability at the Ada source code level. This standard is intended to be used by both application developers and system implementors.

This standard is intended to be compatible with implementations of both the original Ada language standard (ISO/IEC 8652: 1987), and the 1995 revision to that standard (ISO/IEC 8652: 1995).

This standard is comprised of three major components:

- Definitions for terminology and concepts, and definitions and specifications that govern program structures, language-system interaction, and related requirements.

¹Language-independent definitions of these standards were once under development, but work on those projects has been suspended.

- 1 — Definitions of the specific Ada interfaces to the system services defined by the
2 base POSIX standard, presented in the form of Ada packages.
- 3 — Interpretations of Ada semantics with respect to the base standards.

4 The following areas are outside the scope of this standard:

- 5 (1) User interface (shell) and commands associated with Ada program development.
- 6 (2) Ada bindings to the archive/interchange file formats for *tar* and *cpio*.
- 7 (3) Network protocols.
- 8 (4) Graphics and windowing interfaces.
- 9 (5) Database management system interfaces.
- 10 (6) Object or binary code portability.
- 11 (7) System configuration and resource availability.
- 12 (8) Interfaces to the Ada runtime system.

13 This standard describes the external characteristics and facilities that are of impor-
14 tance to applications developers, rather than the implementation approaches that
15 may be employed to achieve them. Special emphasis is placed on those facilities and
16 capabilities needed for the broad spectrum of applications.

17 This standard has been defined exclusively at the source code level. The objective is
18 that a Strictly Conforming POSIX.5 Application can be compiled to execute on any
19 conforming implementation, within the portability of the application Ada code itself.

20 1.2 Normative References

21 The following standards contain provisions that, through reference in this text, con-
22 stitute provisions of this standard. At the time of publication, the editions indicated
23 were valid. All standards are subject to revision, and parties to agreements based
24 on this standard are encouraged to investigate the possibility of applying the most
25 recent editions of the standards indicated below. Members of IEC and ISO maintain
26 registers of currently valid International Standards.

- 27 {1} ISO/IEC 8652: 1995, *Information technology— Programming languages— Ada*
28 [Revision of first edition (ISO 8652:1987)], 15 February 1995. Hereinafter, this
29 is referred to as Ada RM {1}¹.
- 30 {2} ISO/IEC 9945-1: 1996 (IEEE Std 1003.1-1996), *Information technology— Port-*
31 *able Operating System Interface (POSIX)— Part 1: System Application Program*
32 *Interface (API) [C language]*. Hereinafter, this is referred to as POSIX.1 {2}².
- 33 {3} IEEE Standard 1003.1b-1993, *Information Technology— Portable Operating*
34 *System Interface (POSIX)— Part 1: System Application Program Interface (API)*
35 *[C Language]— Amendment 1: Realtime Extension*. Hereinafter, this is referred
36 to as POSIX.1b {3}².

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

39 ²This standard is available from the IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway,
40 NJ 08855-1331, USA. Telephone: 1 (800) 678-IEEE or +1 (908) 981-1393 (outside US).

{4} IEEE Standard 1003.1c-1995, *Information Technology— Portable Operating System Interface (POSIX)— Part 1: System Application Programming Interface (API)— Amendment 2: Threads Extension [C Language]*. Hereinafter, this is referred to as POSIX.1c {4}².

{5} IEEE Standard 1003.1i-1995, *Information Technology— Portable Operating System Interface (POSIX)— Part 1: System Application Programming Interface (API)— Amendment: Technical Corrigenda to Realtime Extension [C Language]*. Hereinafter, this is referred to as POSIX.1i².

NOTE: Items 2, 3, 4, and 5 are the base standards.

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Requirements

A conforming implementation shall meet all of the following criteria:

(1) The system shall support all required interfaces defined within this standard. These interfaces shall support the functional behavior described herein.

(2) For packages and subprograms that are defined to be dependent on implementation options, either the runtime behavior shall be as defined by this standard or references to the name shall be rejected at compile or link time.

(3) The system may provide additional facilities not required by this standard. Nonstandard extensions shall be identified as such in the system documentation. Nonstandard extensions, when used, may change the behavior of functions or facilities defined by this standard, but only if the application *activates an extension*, by reference to an implementation defined extension package in a **with** clause, calling a subprogram in an implementation defined extension package, or using an implementation defined configuration pragma [Ada RM {1} 10.1.5 (8)].

The conformance document shall define an environment in which an application can be run with the behavior specified by this standard. In no case except package name conflicts shall such an environment require modification of a Strictly Conforming POSIX.5 Application. An implementation shall not change package specifications in this standard except by the following:

(a) Adding **with** clauses, pragmas, representation specifications, and comments.

(b) Replacing instances of the words *implementation-defined* with appropriate value(s).

(c) Optionally changing the type `POSIX_Character` (see §2.4.2).

(d) Adding or changing private parts.

(e) Making any other changes that are lexically transparent to Ada compilers.

²This standard is available from the IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. Telephone: 1 (800) 678-IEEE or +1 (908) 981-1393 (outside US).

- 1 (f) Adding specific declarations for which permission is explicitly granted in
2 the text of this standard.
- 3 (g) Changing packages with names of the form POSIX_XXX to child packages of
4 the package POSIX with names of the form POSIX.XXX, and adding library-
5 level renaming declarations of the form:
6 **with** POSIX.XXX;
7 **package** POSIX_XXX **renames** POSIX.XXX;
- 8 (h) Where there is a subprogram which has a parameter with a default value
9 expression in the final position, the default value expression may be omitted
10 and an equivalent overloaded definition of the same function name, without
11 the last parameter, may be provided instead, if this results in a gain in
12 efficiency. For example,
13 **procedure** P(X: T:= E);
14 may be replaced by
15 **procedure** P(X: T);
16 **procedure** P;
17 Calling the version of procedure P without the parameter X should be equiv-
18 alent to calling P(X=>E).
- 19 (4) Only the following types may be implemented using Ada access types:
20 — POSIX.POSIX_String_List
21 — POSIX_Asynchronous_IO.AIO_Descriptor
22 — POSIX_Condition_Variables.Attributes
23 — POSIX_Condition_Variables.Condition
24 — POSIX_Condition_Variables.Condition_Descriptor
25 — POSIX_Message_Queues.Message_Queue_Descriptor
26 — POSIX_Mutexes.Attributes
27 — POSIX_Mutexes.Mutex
28 — POSIX_Mutexes.Mutex_Descriptor
29 — POSIX_Process_Environment.Environment
30 — POSIX_Process_Primitives.Process_Template
31 — POSIX_Semaphores.Semaphore
32 — POSIX_Semaphores.Semaphore_Descriptor
33 — Any other type specifically defined as an Ada access type in this standard
34 No other types defined in this standard shall be implemented using Ada access
35 types.
- 36 (5) A POSIX implementation shall not raise **Program_Error** on elaboration of a
37 POSIX package, or on execution of a POSIX subprogram, due to elaboration
38 order dependencies in the POSIX implementation.
- 39 (6) Except as explicitly provided for in this standard, **POSIX.POSIX_Error** or **Stan-**
40 **dard.Storage_Error** are the only exceptions that shall be raised by operations
41 declared in this standard.
- 42 (7) Wherever this standard states a requirement on an “implementation,” without
43 any other explicit qualification, the requirement applies to all conforming imple-
44 mentations.

1.3.1.2 Documentation

A conformance document shall be available for an implementation claiming conformance to this standard. The conformance document shall have the same structure as this standard, with the information presented in the equivalently numbered sections, clauses, and subclauses. The conformance document shall not contain information about extended facilities or capabilities outside the scope of this standard.

The conformance document shall contain a statement that indicates the full name, number, and date of the standard that applies. The conformance document may also list software standards approved by ISO/IEC or any ISO/IEC member body that are available for use by a Conforming POSIX.5 Application. Applicable characteristics whose documentation is required by one of these standards, or by standards of government bodies, may also be included.

The document shall describe the contents of the packages `POSIX`, `POSIX_Limits`, `POSIX_Options`, `POSIX_Configurable_File_Limits`, and `POSIX_Configurable_System_Limits` and shall state values, the conditions under which those values may change, and the limits of such variations, if any.

The conformance document shall describe the behavior of the implementation for all implementation defined features defined in this standard. This requirement shall be met by listing these features and providing either a specific reference to the system documentation or providing full syntax and semantics of these features. The conformance document may specify the behavior of the implementation for those features where this standard states that implementations may vary or where features are identified as undefined or unspecified.

No specifications other than those described in this subclause shall be present in the conformance document.

The phrase “shall be documented” in this standard means that documentation of the feature shall appear in the conformance document, as described previously, unless the system documentation is explicitly mentioned.

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

1.3.1.3 Conforming Implementation Options

The symbolic constants in the packages `POSIX`, `POSIX_Limits`, and `POSIX_Options` provide declarations for a range of variation in the standard that is implementation defined. The packages `POSIX_Configurable_System_Limits` and `POSIX_Configurable_File_Limits` contain subprograms that may further constrain the implementation defined ranges.

An implementation is required to define all of the subprograms for all of the operations defined in this standard, including those whose implementation is optional. If an unimplemented feature is used, the exception `POSIX.POSIX_Error` shall be raised, with the error code `POSIX.Operation_Not_Implemented`.

1.3.2 Application Conformance

All applications claiming conformance to this standard shall fall within one of the categories defined in the following subclauses.

1.3.2.1 Strictly Conforming POSIX.5 Application

A Strictly Conforming POSIX.5 Application is an application that requires only the facilities described in this standard and Ada RM {1}. Such an application shall be designed to work for all behaviors and values allowed by this standard, including behaviors that are unspecified or implementation defined. Such applications are permitted to adapt to the availability of facilities whose availability is indicated by the values from the packages POSIX, POSIX_Limits, POSIX_Options, POSIX_Configurable_File_Limits, and POSIX_Configurable_System_Limits.

1.3.2.2 Conforming POSIX.5 Application

There are three categories of conforming POSIX.5 applications:

1.3.2.2.1 ISO/IEC Conforming POSIX.5 Application

An ISO/IEC Conforming POSIX.5 Application is an application that uses only the facilities described in this standard and approved Conforming Language bindings for any ISO or IEC standard. Such an application shall include a statement of conformance that documents all options and limit dependencies and all other ISO or IEC standards used.

1.3.2.2.2 <National Body> Conforming POSIX.5 Application

A <National Body> Conforming POSIX.5 Application differs from an ISO/IEC Conforming POSIX.5 Application in that it also may use specific standards of a single ISO/IEC member body, referred to here as “<National Body>.” Such an application shall include a statement of conformance that documents all options and limit dependencies and all other <National Body> standards used.

1.3.2.2.3 Conforming POSIX.5 Application Using Extensions

A Conforming POSIX.5 Application Using Extensions is an application that differs from a Conforming POSIX.5 Application only in that it uses nonstandard facilities that are consistent with this standard. Such an application shall fully document its requirements for these extended facilities, in addition to the documentation required of a Conforming POSIX.5 Application. A Conforming POSIX.5 Application Using Extensions shall be either an ISO/IEC Conforming POSIX.5 Application Using Extensions or a <National Body> Conforming POSIX.5 Application Using Extensions. (See §1.3.2.2.1 and §1.3.2.2.2.)

Section 2: Terminology and General Requirements

2.1 Editorial Conventions

This standard uses the following editorial and typographical conventions. A summary of typographical conventions is shown in Table 2.1.

Table 2.1 – Typographical Conventions

Reference	Example
Ada Identifier	POSIX_String_List
Ada Implementation Defined Element	<i>implementation-defined</i>
Ada Reserved Word	generic
C-Language Data Type	<i>long</i>
C-Language Error Number	[EINTR]
C-Language Function	<i>system()</i>
C-Language Global External	<i>errno</i>
C-Language Header	<sys/stat.h>
Conceptual Attribute	Message Length
Citation	[Ada RM {1} 9.1 (c)]
Cross-Reference: Annex	Annex A
Cross-Reference: Clause	§2.3
Cross-Reference: Other Standard	Ada RM {1}
Cross-Reference: Section	Section 2
Cross-Reference: Subclause	§2.3.4, §2.3.4.5, etc.
Environment Variable	PATH
Figure Reference	Figure 7-3
File Name	/tmp
Implementation Limit	Realtime Signals Maximum
Implementation Option	Job Control option
Table Reference	Table 6-1

Long Ada identifiers are occasionally hyphenated for typographic purposes only; none of the identifiers in this standard have names that include a hyphen.

Qualified name notation, with the package name, is generally used for names that are not defined within the section where the name is used. The exceptions to this rule are names that are defined by the Ada language, and certain names defined in the package POSIX that are used so frequently that the fully qualified form would be merely cumbersome (e.g., POSIX_String is used instead of POSIX.POSIX_String).

Numbers within braces, such as in “POSIX.1 {2},” represent cross references to the normative reference’s clause (see §1.2). If the number is preceded by a B, it represents a bibliographic entry (see Annex A). Bibliographic entries are for information only. Defined terms are shown in three styles, depending on context:

- (1) Terms defined in §2.2.1 and §2.2.2 are expressed as subclause titles. Alternative forms of the terms appear in [brackets]. At the conclusion of this type of definition, the designation of a standard within [brackets] indicates that the text of the definition was copied from that standard, with only editorial changes, if any.

(2) The initial appearances of other terms, applying to a limited portion of the text, are in *italics*.

(3) Subsequent appearances of the term are in Roman font.

Filenames and pathnames are shown in **Courier**. When a pathname is shown starting with “**\$HOME/**”, this indicates the remaining components of the pathname are to be related to the directory named by the **HOME** environment variable.

Defined names that are usually in lowercase are never used at the beginning of a sentence or anywhere else that regular English usage would require capitalization.

Parenthetical expressions within normative text also contain normative information. The general typographic hierarchy of parenthetical expressions is:

{ [()] }

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 reference and there is no normative difference between these two cases.

Annexes marked as *normative* are parts of the standard that pose requirements, exactly the same as the numbered sections, but they follow the body of the standard for clarity of exposition. *Informative* annexes are for information only and pose no requirements. The index, and all material preceding page 1 of this standard (the “front matter”) are also only informative.

Notes that appear in a smaller point size and are indented have two different meanings, depending on their location:

— When they are within the normal text of the document, they are the same as footnotes—informative, posing no requirements on implementations or applications.

— When they are attached to tables or figures, they are normative, posing requirements.

Text marked as examples (including the use of “e.g.”) is for information only.

The typographical conventions listed here are for ease of reading only. Any inconsistencies in the use of typography are unintentional and have no normative meaning in this standard.

2.2 Definitions

2.2.1 Terminology

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

2.2.1.1 conformance document: A document provided by an implementor that contains implementation details as described in §1.3.1.2.

2.2.1.2 implementation defined: An indication that the implementation shall define and document the requirements for correct program constructs and correct data regarding a value or behavior.

1 **2.2.1.3 may:** An indication of an optional feature.

2 (1) With respect to implementations, the word *may* is to be interpreted as an op-
3 tional feature that is not required in this standard but can be provided.

4 (2) With respect to Strictly Conforming POSIX.5 Applications, the word *may* means
5 that the optional feature shall not be used.

6 **2.2.1.4 obsolescent:** An indication that a certain feature may be considered for
7 withdrawal in future revisions of this standard. Use of obsolescent features is depre-
8 cated.

9 NOTE: Obsolescent features are retained for upward compatibility from POSIX.5 {B3},
10 or to maintain symmetry with certain aspects of the C-language binding to POSIX.1 {2}.
11 Their use in Ada applications is discouraged.

12 **2.2.1.5 shall:** An indication of a requirement on the implementation or on Strictly
13 Conforming POSIX.5 Applications, where appropriate.

14 **2.2.1.6 should:**

15 (1) With respect to implementations, an indication of an implementation recommen-
16 dation, but not a requirement.

17 (2) With respect to applications, an indication of a recommended programming prac-
18 tice for applications and a requirement for Strictly Conforming POSIX.5 Appli-
19 cations.

20 **2.2.1.7 supported:** A condition regarding optional functionality.

21 Certain functionality in this standard is optional, but the interfaces to that function-
22 ality are always required. If the functionality is supported, the interfaces work as
23 specified by this standard (except that they do not raise the exception indicating the
24 unsupported case). If the functionality is not supported, the interface shall raise the
25 exception indicating the unsupported case.

26 **2.2.1.8 system documentation:** All documentation provided with an implemen-
27 tation, except the conformance document.

28 Electronically distributed documents for an implementation are considered part of
29 the system documentation.

30 **2.2.1.9 undefined:** An indication that this standard imposes no portability re-
31 quirements on the use by an application of an indeterminate value or the behavior
32 of that application with erroneous program constructs or erroneous data.

33 Implementations (or other standards) may specify the result of using that value or
34 causing that behavior. An application using such behaviors is using extensions, as
35 defined in §1.3.2.2.3.

2.2.1.10 unspecified: An indication that this standard imposes no portability requirements on applications for correct program constructs or correct data regarding a value or behavior.

Implementations (or other standards) may specify the result of using that value or causing that behavior. An application requiring a specific behavior, rather than tolerating any behavior when using that functionality, is using extensions, as defined in §1.3.2.2.3.

2.2.2 General Terms

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

2.2.2.1 abort deferred operation: An operation that is allowed to complete without being affected by abortion. [Ada RM {1} 9.8 (5)] Certain operations are required by the Ada language to be abort deferred. It is implementation defined whether other operations defined by this standard are abort deferred.

2.2.2.2 abort completion point: “A point at which the execution of an aborted construct must complete”. [Ada RM {1} 9.8 (15)]

2.2.2.3 absolute pathname: See *pathname resolution* in §2.3.11.

2.2.2.4 access mode: A form of access to a file or an attribute of a file indicating the kind of operations that may be performed on the file.

See *file access permissions* in §2.3.7.

2.2.2.5 Ada I/O: The input/output operations defined in Annex A of Ada RM {1} and further defined in Section 8 of this standard.

2.2.2.6 address space: The memory locations that can be referenced by a process. [POSIX.1 {2}]

2.2.2.7 appropriate privileges: An implementation defined means of associating privileges with a process with regard to the subprogram calls and options defined in this standard that need special privileges.

There may be zero or more such means. [POSIX.1 {2}]

2.2.2.8 arm a timer: To start a timer measuring the passage of time, enabling the notification of a process when the specified time or time interval has passed.

2.2.2.9 asynchronous I/O operation: An I/O operation that does not of itself cause the task requesting the I/O to be blocked. This implies that the requesting task and the I/O operation may be running concurrently.

2.2.2.10 asynchronous I/O completion: An asynchronous read or write operation is complete when a corresponding synchronous read or write would have completed and any associated status attributes have been updated.

2.2.2.11 asynchronously generated signal: An occurrence of a signal that is generated by some mechanism external to the task receiving the signal; for example, via a call to `POSIX_Signals.Send_Signal` by another process, or via the keyboard. Being asynchronous is a property of how an occurrence of the signal was generated, and not a property of the signal. All signals may be generated asynchronously. The effect of an asynchronously generated occurrence of a reserved signal (see §3.3.1) on an Ada implementation is undefined.

NOTE: Only asynchronously generated signal occurrences are visible to an Ada application as signals. All signal occurrences that are not generated asynchronously are translated into exceptions.

Whether a signal occurrence is generated asynchronously is not to be confused with whether it is delivered or accepted. The difference between delivery and acceptance of a signal is defined in §3.3.1.

2.2.2.12 background process: A process that is a member of a background process group. [POSIX.1 {2}]

2.2.2.13 background process group: Any process group other than a foreground process group that is a member of a session that has established a connection with a controlling terminal. [POSIX.1 {2}]

2.2.2.14 block special file: A file that refers to a device.

A block special file is normally distinguished from a character special file by providing access to the device in a manner such that the hardware characteristics of the device are not visible. [POSIX.1 {2}]

2.2.2.15 blocking behavior: The effect on other tasks in the same partition when a task is blocked by a POSIX operation. Certain POSIX operations are required to block the calling task under defined conditions. For implementation defined reasons a blocked task may prevent other tasks from executing. Names for blocking behaviors are defined in §2.4.1.5.

2.2.2.16 blocked task: An Ada task that is not running or ready to run, as defined in [Ada RM {1} 9 (10)]. A task is either blocked or ready to run. While ready, a task competes for the available execution resources that it requires to run. An operation that causes a task to become blocked is said to *block* the task, and an operation that causes a task to no longer be blocked is said to *unblock* the task.

2.2.2.17 byte: A byte is the measurement unit for data used in this standard. By common usage the term *byte* usually refers to eight bits of data, but within the context of this standard the size of a byte is implementation defined subject to the constraints given in §2.4.1.4. The size of a byte is given by the constant `POSIX._Byte_Size`.

NOTE: In the context of serial I/O (e.g., see Section 7), transmitting a byte of data may require the transmission of more bits than the size of a byte in memory, since for example stop bits and parity bits might be included.

2.2.2.18 canonical input processing: The processing of terminal input in the form of text lines. (See §7.1.1.6.)

2.2.2.19 character: A sequence of one or more values of type `POSIX.POSIX_` - Character.

NOTE: This is the definition of the term *character* when it is used by itself. It does not apply to qualified phrases containing the word *character*, such as “POSIX character” (see §2.2.2.95), “graphic character” (see §2.2.2.58), “Ada character” (see §2.3.3), and “character special file” (see §2.2.2.20).

2.2.2.20 character special file: A file that refers to a device.

One specific type of character special file is a terminal device file, whose access is defined in §7.1. Other character special files have no structure defined by this standard, and their use is unspecified by this standard. [POSIX.1 {2}]

2.2.2.21 child process: See *POSIX process* in §2.2.2.97.

2.2.2.22 clock: An object that measures the passage of time. The current value of the time measured by a clock can be queried and, possibly, set to a value within the legal range of the clock. [POSIX.1b {3}]

2.2.2.23 completion of a call: The execution of a construct or entity is complete when the end of that execution has been reached, or when a transfer of control causes it to be abandoned. Completion due to reaching the end of execution, or due to the transfer of control of an **exit**, **return**, **goto**, **requeue**, or of the selection of a **terminate** alternative is normal completion. Completion is abnormal when control is transferred out of a construct due to abort or the raising of an exception. [Ada RM- {1} 7.6.1 (2)]

2.2.2.24 condition variable: A synchronization object which allows a task to become blocked, until it is unblocked by some event. The unblocking may occur spontaneously, or as a result of a timeout or another task performing a “signal” operation on the condition variable. In use, condition variables are always associated with mutexes. (See §2.2.2.73.)

2.2.2.25 controlling process: The session leader that established the connection to the controlling terminal.

Should the terminal subsequently cease to be a controlling terminal for this session, the session leader shall cease to be the controlling process. [POSIX.1 {2}]

2.2.2.26 controlling terminal: A terminal that is associated with a session.

Each session may have at most one controlling terminal associated with it, and a controlling terminal is associated with exactly one session. Certain input sequences from the controlling terminal (see §7.1) cause signals to be sent to all processes in the process group associated with the controlling terminal. [POSIX.1 {2}]

2.2.2.27 current working directory: See *working directory* in §2.2.2.152.

1 **2.2.2.28 device:** A computer peripheral or an object that appears to the applica-
2 tion as such. [POSIX.1 {2}]

3 **2.2.2.29 directory:** A file that contains directory entries. [POSIX.1 {2}]

4 No two entries in a directory shall have the same filename.

5 **2.2.2.30 directory entry [link]:** A relation that associates a filename with a file.
6 [POSIX.1 {2}]

7 Several directory entries can associate different filenames with the same file.

8 **2.2.2.31 disarm a timer:** To stop a timer from measuring the passage of time,
9 thereby disabling any future process notifications (until the timer is armed again).
10 [POSIX.1b {3}]

11 **2.2.2.32 dot:** The filename consisting of a single dot character (.).

12 See *pathname resolution* in §2.3.11. [POSIX.1 {2}]

13 **2.2.2.33 dot-dot:** The filename consisting of (.).

14 See *pathname resolution* in §2.3.11. [POSIX.1 {2}]

15 **2.2.2.34 effective group ID:** An attribute of a process that is used in determin-
16 ing various permissions, including the file access permissions, described in §2.3.7.
17 [POSIX.1 {2}]

18 See *group ID* in §2.2.2.59. This value is subject to change during the process lifetime,
19 as described in packages `POSIX_Process_Identification` (see §4.1) and `POSIX_-`
20 `Process_Primitives` (see §3.1).

21 **2.2.2.35 effective user ID:** An attribute of a process that is used in determining
22 various permissions, including file access permissions. [POSIX.1 {2}]

23 See *user ID* in §2.2.2.150. This value is subject to change during the process lifetime,
24 as described in packages `POSIX_Process_Identification` (see §4.1) and `POSIX_-`
25 `Process_Primitives` (see §3.1).

26 **2.2.2.36 empty directory:** A directory that contains, at most, entries for dot and
27 dot-dot. [POSIX.1 {2}]

28 **2.2.2.37 empty string [null string]:** A zero-length array whose components are
29 of some character type.

30 **2.2.2.38 environment task:** The anonymous task whose execution elaborates
31 the library items of the declarative part of an active partition, and then calls the
32 main subprogram, if there is one. [Ada RM {1} 10.2 (8)]

33 **2.2.2.39 epoch:** A base reference time defined as 0 hours, 0 minutes, 0.0 seconds,
34 1 January 1970, Universal Coordinated Time{B2}.

2.2.2.40 erroneous execution: The term *erroneous execution* is used in this standard as defined in [Ada RM {1}1.1.5].

2.2.2.41 error code of a task: An attribute of a task, that ordinarily specifies information about the most recent error that caused `POSIX_Error` to be raised. (See §2.4.6.)

2.2.2.42 fifo special file [FIFO]: A type of file with the property that data written to such a file is read on a first-in-first-out basis. [POSIX.1 {2}]

Other characteristics of FIFOs are described in packages `POSIX_Files` (see §5.2) and `POSIX_IO` (see §6.1).

2.2.2.43 file: An object that can be written to, or read from, or both.

A file has certain attributes, including access permissions and type. File types include regular file, character special file, block special file, FIFO special file, and directory. Other types of files may be defined by the implementation. [POSIX.1 {2}]

2.2.2.44 file description: See *open file description* in §2.2.2.79.

2.2.2.45 file descriptor: A per-process unique non-negative integer value used to identify an open file for the purpose of file access. [POSIX.1b {3} 2.2.2.37]

2.2.2.46 file group class: A property of a file indicating access permissions for a process related to the group identification of the process.

A process is in the file group class of a file if the process is not in the file owner class and if the effective group ID or one of the supplementary group IDs of the process matches the group ID associated with the file. Other members of the class may be implementation defined. [POSIX.1 {2}]

2.2.2.47 filename: A nonempty string that is used to name a file. A filename consists of, at most, `POSIX_Limits.Filename_Maxima` 'Last components of type `POSIX.POSIX_Character`. The characters composing the name may be selected from the set of all the character values excluding the slash character and the null character. The filenames dot and dot-dot have special meaning; see *pathname resolution* in §2.3.11. A filename is sometimes referred to as a pathname component. [POSIX.1 {2}]

2.2.2.48 file offset: The byte position in the file where the next I/O operation begins. [POSIX.1 {2}]

Each open file description associated with a regular file, block special file, or directory has a file offset. A character special file that does not refer to a terminal device may have a file offset. There is no file offset specified for a pipe or FIFO.

2.2.2.49 file other class: A property of a file indicating access permissions for a process related to the user and group information of the process.

A process is in the file other class of a file if the process is not in the file owner class or file group class. [POSIX.1 {2}]

2.2.2.50 file owner class: A property of a file indicating access permissions for a process related to the user identification of the process.

A process is in the file owner class of a file if the effective user ID of the process matches the user ID of the file. [POSIX.1 {2}]

2.2.2.51 file permission: Information about a file that is used, along with other information, to determine if a process has read, write, or execute/search permission to a file. [POSIX.1 {2}]

The file permission information is divided into three parts: owner, group, and other. Each part is used with the corresponding file class of processes. These permissions are contained in the *file mode*, as described in the package `POSIX_Permissions` (see §5.1). The detailed usage of the file permission information in access decisions is described in *file access permissions* in §2.3.7.

2.2.2.52 file serial number: A per-file-system unique value for a file.

File serial numbers are unique throughout a file system. [POSIX.1 {2}]

2.2.2.53 file system: A collection of files, together with certain of their attributes. [POSIX.1 {2}]

Each file system provides a separate binding of file serial numbers to files. A given file serial number is associated with at most one file in a file system, but it may refer to distinct files in distinct file systems. That is, each file system defines a new *name space*, giving meaning to the *names* (file serial numbers) that designate files.

2.2.2.54 first open, of a file: Occurs when a process opens a file, message queue, or shared memory object that is not currently open within any process.

2.2.2.55 foreground process: A process that is a member of a foreground process group. [POSIX.1 {2}]

2.2.2.56 foreground process group: A group of processes that have certain privileges, denied to processes in background process groups, when accessing their controlling terminal.

Each session that has established a connection with a controlling terminal has exactly one process group of the session as the foreground process group of that controlling terminal. See §7.1.1.4. [POSIX.1 {2}]

2.2.2.57 foreground process group ID: The process group ID of the foreground process group. [POSIX.1 {2}]

2.2.2.58 graphic character: A graphic character is a sequence of one or more `POSIX.POSIX_Characters` representing a single graphic symbol.

1 **2.2.2.59 group ID:** A value identifying a group of system users.

2 Each system user is a member of at least one group. A group ID is defined in the
3 package `POSIX_Process_Identification` (see §4.1). When the identity of a group
4 is associated with a process, a group ID value is referred to as a real group ID, an
5 effective group ID, one of the (optional) supplementary group IDs, or an (optional)
6 saved set-group-ID.

7 **2.2.2.60 job control:** A facility that allows users to stop (suspend) selectively
8 the execution of processes and continue (resume) their execution at a later time.
9 [POSIX.1 {2}]

10 The user typically employs this facility via the interactive interface jointly supplied
11 by the terminal I/O driver and a command interpreter. Conforming implementations
12 may optionally support job control facilities. The presence of this option is indicated
13 to the application at compile time by the subtype `Job_Control_Support` in pack-
14 age `POSIX_Options` or at run time by the value returned by the function `Job_Con-`
15 `trol_Is_Supported` in package `POSIX_Configurable_System_Limits`; see §2.4
16 and §4.5.

17 **2.2.2.61 last close, of a file:** The act of a process closing a file, message queue,
18 or shared memory object that results in the file, message queue, or shared memory
19 object no longer being open within any process.

20 **2.2.2.62 link:** See *directory entry* in §2.2.2.30.

21 **2.2.2.63 link count:** The number of directory entries that refer to a particular
22 file. [POSIX.1 {2}]

23 **2.2.2.64 login:** The unspecified (by this standard) activity by which a user gains
24 access to the system. [POSIX.1 {2}]

25 Each login shall be associated with exactly one login name. [POSIX.1 {2}]

26 **2.2.2.65 login name:** A user name that is associated with a login. [POSIX.1 {2}]

27 **2.2.2.66 map a range of addresses:** To create an association between a range
28 of a process's address space and a range of physical memory or some memory object,
29 such that a reference to an address in that range of the address space results in a
30 reference to the associated physical memory or memory object. The mapped memory
31 or memory object is not necessarily memory-resident. [POSIX.1b {3}]

32 **2.2.2.67 memory object:** Either a file or shared memory object. When used in
33 conjunction with `Map_Memory`, `Open_And_Map_Shared_Memory` or `Open_Or_Cre-`
34 `ate_And_Map_Shared_Memory`, a memory object will appear in the calling process's
35 address space. [POSIX.1b {3}]

36 **2.2.2.68 memory-resident:** Managed by the implementation in such a way as to
37 provide an upper bound on memory access times. [POSIX.1b {3}]

1 **2.2.2.69 message:** Information that can be transferred among tasks (possibly in
2 different processes) by being added to and removed from a message queue.

3 **2.2.2.70 message queue:** An object to which messages can be added and re-
4 moved. Characteristics and interfaces associated with message queues are defined
5 in Section 15.

6 **2.2.2.71 message queue descriptor:** A per-process unique value used to iden-
7 tify an open message queue.

8 **2.2.2.72 mode:** A collection of attributes that specifies the type of a file and its
9 access permissions.

10 See *file access permissions* in §2.3.7. [POSIX.1 {2}]

11 **2.2.2.73 mutex:** A synchronization object used to allow multiple tasks (possibly
12 in different processes) to serialize their access to shared data or other shared re-
13 sources. The name derives from the capability it provides, namely, mutual exclusion.

14 **2.2.2.74 mutex owner:** The task that has locked a mutex becomes its owner and
15 remains the owner until that same task unlocks the mutex.

16 **2.2.2.75 noncanonical input processing:** The processing of terminal input as
17 uninterpreted *characters* (see Section 7).

18 **2.2.2.76 null character:** The value `POSIX_Character'Value(0)`, if defined.

19 The use of a null character in filenames or environment variable names or values
20 produces undefined results. Implementations shall not return null characters to ap-
21 plications in filenames or environment variable names or values.

22 **2.2.2.77 null string:** See *empty string* in §2.2.2.37.

23 **2.2.2.78 open file:** A file that is currently associated with a file descriptor. [PO-
24 SIX.1 {2}]

25 **2.2.2.79 open file description:** A record of how a process or group of processes
26 is accessing a file.

27 Each file descriptor shall refer to exactly one open file description, but an open file
28 description may be referred to by more than one file descriptor. A file offset, file
29 status (see §6.1.7), and file access modes are attributes of an open file description.
30 [POSIX.1 {2}]

31 **2.2.2.80 orphaned process group:** A process group in which the parent of every
32 member is either itself a member of the group or is not a member of the session of
33 the group. [POSIX.1 {2}]

34 The process group is no longer a member of the session of the process that created it.
35 This can occur when all other members of the session exit.

1 **2.2.2.81 page:** The granularity of memory mapping and locking.

2 A fixed-length contiguous range of a process's address space. Physical memory and
 3 memory objects can be mapped into a process's address space on page boundaries
 4 and in integral multiples of pages. Process address space can be locked into memory
 5 (i.e., made memory-resident) on page boundaries and in integral multiples of pages.

6 NOTE: There is no implied requirement that usage of the term *page* in this interface
 7 for memory mapping necessarily be the same as the term might be used in a virtual
 8 memory implementation.

9 **2.2.2.82 page size:** The number of storage units in a page.

10 NOTE: Unlike POSIX.1 {2}, the page size is given in storage units here rather than
 11 bytes. This is for consistency with the units used in the address arithmetic provided by
 12 System.Storage_Elements, in case the system storage unit is not one byte.

13 **2.2.2.83 parent directory:**

14 (A) When discussing a given directory, the directory that contains a directory entry
 15 for the given directory. The parent directory is represented by the pathname
 16 dot-dot in the given directory.

17 (B) When discussing other types of files, a directory containing a directory entry for
 18 the file under discussion.

19 This concept does not apply to dot and dot-dot. [POSIX.1 {2}]

20 **2.2.2.84 parent process:** See *POSIX process* in §2.2.2.97.

21 **2.2.2.85 parent process ID:** An attribute of a new process after it is created by
 22 a currently active process.

23 The parent process ID of a process is the process ID of its creator, for the lifetime of
 24 the creator. After the lifetime of the creator has ended, the parent process ID is the
 25 process ID of an implementation defined system process. [POSIX.1 {2}]

26 **2.2.2.86 partition:** A partition is a program or part of a program that can be
 27 invoked from outside the Ada implementation. Each partition may run in a separate
 28 address space, possibly on a separate computer. An *active partition* is a partition
 29 that contains at least one task. Every active partition has an *environment task*, on
 30 which all the other tasks of that partition depend. [Ada RM {1} 10.2 (2)] An active
 31 partition corresponds to a POSIX process.

32 **2.2.2.87 path prefix:** A pathname, with an optional ending slash, that refers to
 33 a directory. [POSIX.1 {2}]

34 **2.2.2.88 pathname:** A nonempty string that is used to identify a file. A path-
 35 name consists of, at most, POSIX_Limits.Pathname_Maxima'Last components of
 36 type POSIX.POSIX_Character. It has an optional beginning slash followed by zero
 37 or more filenames separated by slashes. If the pathname refers to a directory, it may
 38 also have one or more trailing slashes. Multiple successive slashes are considered

the same as one slash. A pathname that begins with two successive slashes may be interpreted in an implementation defined manner, although more than two leading slashes shall be treated exactly the same as a single slash. The interpretation of the pathname is described in §2.3.11.

2.2.2.89 pathname component: See *filename* in §2.2.2.47.

2.2.2.90 permission: See *file permission* in §2.2.2.51.

2.2.2.91 persistence: A characteristic of semaphores, shared memory, and message queues requiring that the object and its state (including data, if any) are preserved after last close (the object is no longer referenced by any process). Persistence of an object does not necessarily imply that the state of the object is maintained across a system crash or a system reboot.

2.2.2.92 pipe: An object accessed by one of the pair of file descriptors created by the `POSIX_IO.Create_Pipe` procedure.

Once created, the file descriptors can be used to manipulate a pipe, and it behaves identically to a FIFO special file when accessed in this way. It has no name in the file hierarchy.

2.2.2.93 portable filename character set: The set of characters from which portable filenames are constructed.

For a filename to be portable across conforming implementations of this standard, it shall consist only of the following characters:

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	.	_	-													

The last three characters are the period, underscore, and hyphen characters, respectively. The hyphen shall not be used as the first character of a portable filename. Upper- and lowercase letters shall retain their unique identities between conforming implementations.

2.2.2.94 portable pathname character set: The set of characters from which portable pathnames are constructed. The set contains all the characters of the portable filename set, plus the character slash (/).

2.2.2.95 posix Character: A value of the type `POSIX_Character`. An array of POSIX characters, of type `POSIX_String` is called a *POSIX string*.

2.2.2.96 posix I/O: The input/output operations defined by this standard, except for Section 8.

2.2.2.97 posix process: A conceptual object, having an associated address space, one or more threads of control executing within that address space, a collection of system resources required for execution, and certain other attributes. (See §2.3.1.) A

1 POSIX process is said to perform an action if any of the conceptual threads of control
2 within it performs the action.

3 Many of the *system resources* defined by this standard are shared among all of the
4 threads of control within a process. These include (but are not limited to): the process
5 ID; the parent process ID; process group ID; session membership; real, effective, and
6 saved set-user-ID; real, effective, and saved set-group-ID; supplementary group IDs;
7 current working directory; root directory; file mode creation mask; file descriptors;
8 timers; file locks; mutexes; environment variables; process argument list; possibly, a
9 process signal mask. (See §3.3.1.)

10 A process is created by another process with procedures `POSIX_Process_Primitives.Start_Process`,
11 `POSIX_Process_Primitives.Start_Process_Search`, or the function `POSIX_Unsafe_Process_Primitives.Fork`. The process that is-
12 sues `Start_Process`, `Start_Process_Search`, or `Fork` is known as the parent
13 process, and the newly created process is the child process.

15 **2.2.2.98 potentially blocking operation:** An operation that is not allowed
16 within a protected action, because it may be required to block the calling task.
17 [Ada RM {1} 9.5.1 (8)] Certain operations are defined by the Ada language to be po-
18 tentially blocking. In addition, every operation defined by this standard whose effect
19 is defined as blocking the calling task under any circumstance is also a potentially
20 blocking operation.

21 **2.2.2.99 priority:** The general term for an integer-valued attribute of processes,
22 tasks, messages, and asynchronous I/O operations, whose value is used in selecting
23 among entities of the same kind. Numerically higher values represent higher priori-
24 ties, and are given preference for selection over lower priorities.

25 The *priority of an Ada task* is an integer that indicates a degree of urgency and is
26 the basis for resolving competing demands of tasks for resources. Unless otherwise
27 specified, whenever tasks compete for processors or other implementation defined
28 resources, the resources are allocated to the task with the highest priority value.
29 The *base priority* of a task is the priority with which it was created, or to which it
30 was later set by `Dynamic_Priorities.Set_Priority` (see [Ada RM {1} D.5]). At all
31 times, a task also has an *active priority*, which generally reflects its base priority as
32 well as any priority it inherits from other sources. *Priority inheritance* is the process
33 by which the priority of a task or other entity (e.g., a protected object; see [Ada RM-
34 {1} D.3]) is used in the evaluation of another task's active priority. [Ada RM {1} D.1
35 (15)]

36 At any time, the active priority of a task is the maximum of all the priorities the task
37 is inheriting at that instant. [Ada RM {1} D.1 (20)]

38 **2.2.2.100 privilege:** See *appropriate privileges* in §2.2.2.7.

39 **2.2.2.101 process:** See *POSIX process* in §2.2.2.97.

40 **2.2.2.102 process group:** A collection of processes that permits the signaling of
41 related processes.

1 Each process in the system is a member of a process group that is identified by its
2 process group ID. A newly created process joins the process group of its creator. [PO-
3 SIX.1 {2}]

4 **2.2.2.103 process group ID:** A unique value identifying a process group during
5 its lifetime.

6 A process group ID shall not be reused by the system until the process group lifetime
7 ends. [POSIX.1 {2}]

8 **2.2.2.104 process group leader:** The unique process, within a process group,
9 that created the process group.

10 **2.2.2.105 process group lifetime:** A period of time that begins when a process
11 group is created and ends when the last remaining process in the group leaves the
12 group, due either to the end of the process lifetime of the last process or to the last
13 remaining process calling the Set_Process_Group_ID procedure. [POSIX.1 {2}]

14 **2.2.2.106 process ID:** A unique value identifying a process during its lifetime.

15 The process ID is defined in the package POSIX_Process_Identification. (See
16 §4.1.) A process ID shall not be reused by the system until the process lifetime ends.
17 In addition, if there exists a process group where the process ID of the process group
18 leader is equal to that process ID, that process ID shall not be reused by the system
19 until the process group lifetime ends.

20 An implementation shall reserve a value of process ID for use by system processes.
21 A process that is not a system process shall not have this process ID.

22 **2.2.2.107 process lifetime:** A period of time that begins when a process is cre-
23 ated and ends when its process ID is returned to the system. [POSIX.1 {2}]

24 After a process is created, it is considered active. Its threads of control and address
25 space exist until it terminates. It then enters an inactive state where certain re-
26 sources may be returned to the system, although some resources, such as the process
27 ID, are still in use. When another process executes a Wait_For_Child_Process
28 procedure for an inactive process, the remaining resources are returned to the sys-
29 tem. The last resource to be returned to the system is the process ID. At this time,
30 the lifetime of the process ends. [POSIX.1 {2}]

31 **2.2.2.108 program:** A set of partitions, which can execute in parallel with one
32 another, possibly in a separate address space, and possibly on a separate computer.
33 [Ada RM {1} 10.2 (1)]

34 **2.2.2.109 read-only file system:** A file system that has implementation defined
35 characteristics restricting modifications. [POSIX.1 {2}]

36 **2.2.2.110 ready task:** A task that is not blocked. (See §2.2.2.16.) The ready
37 tasks include those that are running as well as those that are waiting for a processor.

NOTE: For consistency, this standard uses the definition of *ready* from Ada RM {1}, which includes running tasks among those that are ready. Thus the meaning of *ready* is close to the meaning of *runnable* as used in POSIX.1c {4}, but the two terms differ in that *ready* includes running threads. This difference in terminology requires differences in several descriptions between this Ada binding and the base POSIX standards, to preserve compatible semantics.

2.2.2.111 real group ID: The attribute of a process that, at the time of process creation, identifies the group of the user who created the process.

See *group ID* in §2.2.2.59. This value is subject to change during the process lifetime, as described in §4.1. [POSIX.1 {2}]

2.2.2.112 real user ID: The attribute of a process that, at the time of process creation, identifies the user who created the process.

See *user ID* in §2.2.2.150. This value is subject to change during the process lifetime, as described in §4.1. [POSIX.1 {2}]

2.2.2.113 referenced shared memory object: A shared memory object that is open or has one or more mappings defined on it. [POSIX.1b {3}]

2.2.2.114 region: As relates to the address space of a process, a sequence of addresses.

As relates to a file, a sequence of offsets.

2.2.2.115 regular file: A file that is a randomly accessible sequence of bytes, with no further structure imposed by the system.

2.2.2.116 relative pathname: See *pathname resolution* in §2.3.11.

2.2.2.117 reserved signal: Signals which the application cannot accept, and for which the application cannot modify the signal action or masking, because the signals are reserved for use by the Ada language implementation. The reserved signals defined by this standard are `Signal_Abort`, `Signal_Alarm`, `Signal_Floating_Point_Error`, `Signal_Illegal_Instruction`, `Signal_Segmentation_Violation`, `Signal_Bus_Error`. If the implementation supports any signals besides those defined by this standard, the implementation may also reserve some of those.

2.2.2.118 resolution, time: The minimum time interval that a clock can measure or whose passage a timer can detect. [POSIX.1b {3}]

2.2.2.119 root directory: A directory, associated with a process, that is used in pathname resolution for pathnames that begin with a slash. [POSIX.1 {2}]

2.2.2.120 running task: The task currently being executed by a processor.

2.2.2.121 saved set-group-ID: When the `Saved_IDs_Support` option is implemented, an attribute of a process that allows some flexibility in the assignment of the effective group ID attribute, as described for `Set_Group_ID`.

See §2.3.7 and Section 4.

2.2.2.122 saved set-user-ID: When the `Saved_IDs_Support` option is implemented, an attribute of a process that allows some flexibility in the assignment of the effective user ID attribute, as described for `Set_User_ID`.

See §2.3.7 and Section 4.

2.2.2.123 semaphore: A shareable resource that has a non-negative integer value. When the value is zero, there is a (possibly empty) set of tasks awaiting the availability of the semaphore.

2.2.2.124 semaphore decrement operation: An operation that decrements the value of a semaphore, blocking until this is possible. If, prior to the operation, the semaphore's value is zero, the semaphore decrement operation shall cause the calling task to be blocked and added to the set of tasks (possibly in different processes) awaiting the semaphore. Otherwise, the semaphore value is decremented. (See §11.1.7.)

2.2.2.125 semaphore increment operation: An operation that increments the value of a semaphore, or unblocks a waiting task.

If, prior to the operation, there are any tasks awaiting the semaphore, then some task from that set shall be removed from the set and be unblocked. Otherwise, the semaphore value shall be incremented. (See §11.1.8.)

2.2.2.126 session: A collection of process groups established for job control purposes. [POSIX.1 {2}]

Each process group is a member of a session. A process is considered to be a member of the session of which its process group is a member. A newly created process joins the session of its creator. A process can alter its session membership by the procedure `Create_Session` in the package `POSIX_Process_Identification`. Implementations that support `Set_Process_Group_ID` can have multiple process groups in the same session. [POSIX.1 {2}]

2.2.2.127 session leader: A process that has created a session. (See §4.1.2.)

2.2.2.128 session lifetime: The period between when a session is created and the end of the lifetime of all the process groups that remain as members of the session. [POSIX.1 {2}]

2.2.2.129 shared memory object: An object that represents memory that can be mapped concurrently into the address space of more than one process. These are named regions of storage which may be independent of the file system and can be mapped into the address space of one or more processes to allow them to share the associated memory.

2.2.2.130 signal: A mechanism by which a process may be notified of, or affected by, an event occurring in the system.

Examples of such events include hardware exceptions and specific actions by processes. The term *signal* is also used to refer to the event itself. [POSIX.1 {2}]

2.2.2.131 signal queueing: When queueing is enabled for a signal, occurrences of that signal are queued in FIFO order and information is included if the signal is from a source that supplies information. Otherwise, the signal queue may be only one occurrence deep and it is implementation defined whether the data is included. Support for signal queueing is governed by the Realtime Signals option. Not all signals may support queueing.

2.2.2.132 slash: The literal character “/”.

This character is also known as *solidus* in ISO 8859-1 {B4} [POSIX.1 {2}]

2.2.2.133 storage unit: The length of an addressable element of storage in the machine, measured in bits. (Every storage element has the same size.) This term is used here as defined in [Ada RM {1} 13.7 (31)], where it is declared as `System.Storage_Unit`.

NOTE: The storage unit is very likely to be one byte, but this is not a requirement. For example, it might be 32 or 64 bits.

2.2.2.134 successfully transferred: Data is successfully transferred for a write operation to a regular file when the system ensures that all data written is readable on any subsequent open of the file (even one that follows a system or power failure) in the absence of a failure of the physical storage medium. Data is successfully transferred for a read operation when an image of the data on the physical storage medium is available to the requesting task.

2.2.2.135 supplementary group ID: An attribute of a process, used in determining file access permissions.

A process has group IDs in addition to the effective group ID. The size of this list of supplementary group IDs is specified at compile time by `Groups_Maxima` in package `POSIX_Limits`, or at run time by the value of the function `Groups_Maximum` in package `POSIX_Configurable_System_Limits`. The supplementary group IDs of a process are set to the supplementary group IDs of the parent process when the process is created. Whether the effective group ID of a process is included in or omitted from its list of supplementary group IDs is unspecified. [POSIX.1 {2}]

2.2.2.136 synchronized I/O completion: The state of an I/O operation that has either been successfully transferred or diagnosed as unsuccessful. [POSIX.1b {3}]

2.2.2.137 synchronized I/O data integrity completion: A degree of completion for an I/O operation that occurs when:

- (1) For reads, the operation has been completed or diagnosed as unsuccessful. The read is complete only when an image of the data has been successfully transferred to the requesting task. If there were any pending write requests affecting the data to be read at the time that the synchronized read operation was requested, these write requests shall be successfully transferred prior to reading the data.

(2) For writes, the operation has been completed or diagnosed as unsuccessful. The write is complete only when the data specified in the write request is successfully transferred, and all file system information required to retrieve the data is successfully transferred.

File attributes that are not necessary for data retrieval (Last Access Time, Last Modification Time, Last Status Change Time) need not be successfully transferred prior to returning to the calling task. [POSIX.1b {3}]

2.2.2.138 synchronized I/O file integrity completion: Identical to a synchronized I/O data integrity completion with the addition that all file attributes relative to the I/O operation (including Last Access Time, Last Modification Time, Last Status Change Time) shall be successfully transferred prior to returning to the calling task. [POSIX.1b {3}]

2.2.2.139 synchronized I/O operation: An I/O operation performed on a file that provides the application assurance of the integrity of its data and files. (See §2.2.2.137 and §2.2.2.138.) [POSIX.1b {3}]

2.2.2.140 synchronous I/O operation: An I/O operation that causes the task requesting the I/O to be blocked from further use of the processor until that I/O operation completes. [POSIX.1b {3}]

NOTE: A synchronous I/O operation does not imply synchronized I/O data integrity completion or synchronized I/O file integrity completion.

2.2.2.141 system: An implementation of this standard. [POSIX.1 {2}]

2.2.2.142 system crash: An event initiated by an unspecified circumstance that causes all processes (possibly other than special system processes) to be terminated in an undefined manner, after which any changes to the state and contents of files created or written to by a Conforming POSIX.5 Application prior to the interval are undefined, except as required elsewhere in this standard. [POSIX.1b {3}]

2.2.2.143 system process: An object, other than a process executing an application, that is defined by the system and has a process ID. [POSIX.1 {2}]

An implementation shall reserve at least one process ID for system processes.

2.2.2.144 system reboot: An implementation defined sequence of events that may result in the loss of transitory data, i.e., data that is not saved in permanent storage. This includes message queues, shared memory, semaphores, and processes. [POSIX.1b {3}]

2.2.2.145 task: An Ada object with a thread of control, defined by [Ada RM {1} 9] as follows:

The execution of an Ada program consists of the execution of one or more tasks. Each task represents a separate thread of control that proceeds independently and concurrently between the points where it interacts with other tasks.

2.2.2.146 terminal [terminal device]: A character special file that obeys the specifications of §7.1. [POSIX.1 {2}]

2.2.2.147 thread of control: A sequence of instructions executed by a conceptual sequential subprogram, independent of any programming language. More than one thread of control may execute concurrently, interleaved on a single processor, or on separate processors. The conceptual threads of control in an Ada application are Ada tasks. They may, but need not, correspond to the POSIX threads defined in POSIX.1c {4}.

2.2.2.148 timer: An object that can notify a process when the time as measured by a particular clock has reached or passed a specified value, or when a specified amount of time, as measured by a particular clock has passed. Timers are per process; that is, they cannot be shared between processes.

2.2.2.149 timer overrun: A condition that occurs each time a timer for which there is already an expiration signal queued to the process expires. [POSIX.1b {3}]

2.2.2.150 user ID: A value identifying a system user.

The user ID is defined in the package `POSIX_Process_Identification`. When the identity of a user is associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or an (optional) saved set-user-ID.

2.2.2.151 user name: A value of `POSIX_String` that is used to identify a user, as described in §9.1.

2.2.2.152 working directory [current working directory]: A directory, associated with a process, that is used in pathname resolution for pathnames that do not begin with a slash. [POSIX.1 {2}]

2.2.3 Abbreviations

For the purposes of this standard, the following abbreviations apply, in addition to the abbreviations given for the normative references in §1.2.

2.2.3.1 Ada83: The original Ada language standard, approved by ANSI in 1983 and by ISO/IEC in 1987, item {B?} in Annex A.

2.2.3.2 Ada95: The 1995 Ada language standard, item {1} in §1.2, used in contrast to Ada 83.

2.2.3.3 POSIX.5: IEEE Standard 1003.5-1992, {B3} in Annex A.

2.2.3.4 POSIX.5b: This standard.

2.3 General Concepts

2.3.1 Process/Active Partition Relationship

An Ada active partition in execution shall behave as if it is a single POSIX process, throughout this standard. Thus, all the tasks of the active partition shall execute within the environment of the same POSIX process, and share the same set of process attributes. (See §2.2.2.97.)

NOTE: The active partition in Ada95 (ISO/IEC 8652: 1995) generalizes the concept of a main program execution in Ada83 (ISO/IEC 8652: 1987).

NOTE: While every Ada active partition in a POSIX environment must correspond to some process, the two concepts are not exactly equivalent. In particular, a process may execute a program written in other languages. Moreover, during the lifetime of a process it may execute several different programs (or Ada active partitions), possibly written in different languages, sequentially. (See §3.2.)

2.3.2 Task/Thread Relationship

The only threads of control visible to an Ada application via this standard are tasks; requirements that relate to threads of control are generally described in terms of Ada tasks.

An implementation may optimize away the thread of control of certain tasks (known as *passive tasks*), if one of the following applies:

- The implementation takes responsibility for the safety of this optimization. That is, it is able to determine that it will not violate any requirements of Ada RM {1}, or change the effect of any operations or pragmas defined by this standard that are supported by the implementation and used within the application.
- The application takes responsibility for the safety of this optimization, requesting it explicitly by implementation defined means, such as a pragma.

A process may contain threads of control that do not correspond to Ada tasks. Such threads of control may be created invisibly by the implementation of the Ada language or of this standard, or via operating system interfaces not defined by this standard. Such other threads of control are not required to behave as Ada tasks. The implementation shall not allow the application to refer to such threads as tasks, via task names or values of type `Ada_Task_Identification.Task_ID`.

NOTE: The tasks of an Ada program may also be implemented using more than one process in the underlying system as long as all the process attributes specified by this standard (e.g., see §3.1.2, §3.2.2, and §4) are shared by all the tasks in the program, and all tasks in the Ada active partition appear to the other POSIX processes in the system to be a single POSIX process, when viewed through the interfaces defined in this standard.

2.3.3 Ada Character Differences

The Ada RM {1} defines the type `Character` as “a character type whose values correspond to the 256 code positions of Row 00 (also known as Latin-1) of the ISO 10646 Basic Multilingual Plane (BMP).” It also defines the type `Wide_Character`,

which “is a character type whose values correspond to the 65536 code positions of the ISO 10646 Basic Multilingual Plane (BMP).”

This standard defines its own character set, `POSIX_Character`, which is neither required nor prohibited from having the same set of values and representation as `Standard.Character` or `Standard.Wide_Character`.

2.3.4 Posix Signals Are Not Interrupts

The application program interfaces for signals defined by this standard are synchronous; that is, a task executes an operation that causes it to block until an occurrence of the signal is available. The preferred interface is via the `Await_Signal` and `Await_Signal_Or_Timeout` operations (see §3.3.15 and §3.3.16). For compatibility with POSIX.5 {B3}, a signal can also be attached to an entry of a task, via the Ada interrupt-entry mechanism, if the Signal Entries option is supported. (See §3.3.17.)

NOTE: The use of task entries to accept signals is obsolescent.

2.3.5 System Call Exception Errors

Ada supports an exception mechanism that is used in this binding to report errors from system calls. The details of the error are reported by an error code that can be accessed via the function `Get_Error_Code` in package `POSIX`.

2.3.6 Extended Security Controls

The access control (see §2.3.7) and privilege mechanisms (see §2.2.2.7) have been defined to allow implementation defined *extended security controls*. These permit an implementation to provide security mechanisms to implement different security policies than are described in this standard. These mechanisms shall be implemented and mapped onto the POSIX interfaces in such a way as to appear to be an extension to the POSIX security system.

2.3.7 File Access Permissions

The standard file access control mechanism uses the file permission set, as described below. This permission set is determined at file creation by the procedures `POSIX_IO.Open_Or_Create`, `POSIX_Files.Create_FIFO`, or `POSIX_Files.Create_Directory` and are changed by `POSIX_Files.Change_Permissions`. The permission set can be read by the function `POSIX_File_Status.Permission_Set_Of`.

Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An additional access control mechanism shall only further restrict the access permissions defined by the file permission set. An alternate access control mechanism shall

- (1) Specify file permission set for the owner class, group class, and other class of the file, corresponding to the access permissions.
- (2) Be enabled only by explicit user action, on a per-file basis by the file owner or a user with the appropriate privilege.

- (3) Be disabled for a file after the file permission set is changed for that file. The disabling of the alternate mechanism need not disable any additional mechanisms defined by an implementation.

Whenever a process requests file access permission for read, write, or execute/search, if no additional mechanism denies access, access is determined as follows:

- (1) If a process has the appropriate privilege:
 - (a) If read, write, or directory search permission is requested, access is granted.
 - (b) If execute permission is requested, access is granted if execute permission is granted to at least one user by the file permission set or by an alternate access control mechanism; otherwise, access is denied.
- (2) Otherwise:
 - (a) The file permission set of a file contains read, write, and execute/search permissions for the file owner class, file group class, and file other class.
 - (b) Access is granted if an alternate access control mechanism is not enabled and the requested access permission is set for the class to which the process belongs, or if an alternate access control mechanism is enabled and it allows the requested access; otherwise, access is denied.

Unless otherwise stated, the user ID is the effective user ID of the process and the group ID is the effective group ID of the process. The effective user ID and the effective group ID can be saved in the saved set-user-ID or saved set-group-ID if the Saved IDs option is supported. (See §4.1.3.)

2.3.8 File Hierarchy

Files in the system are organized in a hierarchical structure in which all of the non-terminal nodes are directories and all of the terminal nodes are any other type of file. Because multiple directory entries may refer to the same file, the hierarchy is properly described as a *directed graph*.

2.3.9 Filename Portability

Filenames should be constructed from the portable filename character set because the use of other characters can be confusing or ambiguous in certain contexts. The function `Is_Portable_Filename` (see §2.4) is provided for checking a `POSIX_String` to verify if it is a portable filename. The function `Is_Portable_Pathname` (see §2.4) is provided to check for portable pathnames.

2.3.10 File Times Update

Each file has three associated time value attributes. These are the Last Access Time, the Last Modification Time, and the Last Status Change Time. They are updated when file data has been accessed, file data has been modified, or file status has been changed, respectively. They are accessed via functions defined in the package `POSIX_File_Status`.

For each function or procedure in this standard that reads or writes file data or changes the file status, the appropriate attributes are noted as *marked for update*.

If an implementation of such a function marks for update a time-related attribute not specified by this standard, this shall be documented, except that any changes caused by pathname resolution need not be documented. For the other functions in this standard (those that are not explicitly required to read or write file data or change file status, but that in some implementations happen to do so), the effect is unspecified.

An implementation may update attributes that are marked for update immediately, or it may update such attributes periodically. When the attributes are updated, they are set to the current time and the update marks are cleared. All attributes that are marked for update shall be updated when the file is no longer open by any process, or when a `POSIX_File_Status.Get_File_Status` operation is performed on the file. Other times when updates are done are unspecified. Updates are not done for files on read-only file systems.

2.3.11 Pathname Resolution

Pathname resolution is performed for a process to resolve a pathname to a particular file in a file hierarchy. There may be multiple pathnames that resolve to the same file. (See §5.2.1 and §5.2.3.)

Each filename in the pathname is located in the directory specified by its predecessor (for example, in the pathname fragment “a/b”, file “b” is located in directory “a”). Pathname resolution fails if this cannot be accomplished. If the pathname begins with a slash, the predecessor of the first filename in the pathname is taken to be the root directory of the process (such pathnames are referred to as absolute pathnames). If the pathname does not begin with a slash, the predecessor of the first filename of the pathname is taken to be the current working directory of the process (such pathnames are referred to as relative pathnames). If the pathname begins with two leading slashes, the predecessor of the first filename in the pathname may be interpreted in an implementation defined manner.

The interpretation of a pathname component is dependent on `Filename Limit` and whether the `Filename Truncation` option is supported for the path prefix of the component. If the length in POSIX characters of a pathname component exceeds `Filename_Limit` and the `Filename Truncation` option is not supported for the pathname prefix, the implementation shall raise the exception `POSIX_Error` and set the error code `Filename_Too_Long`. Otherwise, the implementation shall ignore all the characters after the initial substring whose length in POSIX characters is the value returned by `Filename_Limit` for the pathname component. (See Section 5.)

The special filename, dot (`.`), refers to the directory specified by its predecessor. The special filename, dot-dot (`..`), refers to the parent directory of the directory specified by its predecessor. As a special case, in the root directory, dot-dot may refer to the root directory itself.

A pathname consisting of a single slash (`/`) resolves to the root directory of the process. A null pathname is invalid.