STANDARDS PROJECT

Draft Standard for Information Technology— POSIX Ada Language Interfaces— Part 2: Binding for Realtime Extensions

Sponsor

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

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 to the application program interfaces (API) defined in P1003.4 and P1003.4a.

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

P1003.20/D1.1
December 1992

Copyright ©1992 by the
Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved.

This is an unapproved and unpublished IEEE Standards Draft, subject to change. The publication, distribution, or copying of this draft, as well as all derivative works based on this draft, is expressly prohibited except as set forth below.

Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activites only, and subject to the restrictions contained herein.

Permission is also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position, subject to the restrictions contained herein.

Permission is hereby also granted to the preceding entities to make limited copes of this document in an electronic form only for the stated activities.

The following restrictions apply to reproducing or transmitting the document in any form: 1) all copies or portions thereof must identify the document's IEEE project number and draft number, and must be accompanied by this entire notice in a prominent location; 2) no portion of this document may be redistributed in any modified or abridged form without the prior approval of the IEEE Standards Department.

Other entities seeking permission to reproduce this document, or any portion thereof, for standardization or other activities, must contact the IEEE Standards Department for the appropriate license.

Use of the information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
+1 (908) 562-3800
+1 (908) 562-1571 [FAX]
November 1992 *SH XXXXX*

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Editor's Notes

This section will not appear in the final document. It is used for editorial comments concerning this draft.

This draft is for mock ballot. A mock ballot is not a formal IEEE ballot. The purpose is to expose any issues that should be addressed before the document enters the formal ballot process, while it is still possible to make major changes.

Anyone is welcome to submit a mock ballot on this document. If you plan to submit a mock ballot it is important that you read the balloting instructions on the cover letter, and submit your comments by the due date.

The present form is a so-called "thin" binding to the P1003.4 C-language interface. Therefore, in order to understand this document you will need a copy of P1003.4 D13. This can be obtained from the IEEE Standards Office. In the U.S.A. call +1(800) 678-IEEE. In Europe, call +1 (908) 981-1393.

Besides the technical content, we are interested in comments on the form of the proposed standard. For example, IEEE TECOS and ISO WG15 have directed us to produce a thin binding to a language-independent specification. Since there is no LIS, we used the C-language interface. Is this acceptable? Should we wait for the P1003.4 LIS to catch up? Or should we proceed with a thick Ada binding, like the P1003.5 document which we are extending? See § 1.4.5 for more on this subject.

A more superficial issue is whether to replicate the text of the package specifications. Compare Section 16 against the others, to see how the full text of the Ada declarations is given in both the package specification and the synopses. In the other sections, elipsis is used in the package specification and the full text is only given once, in the synopses.

Contact Information

The 1003.20 technical editor:

Ted Baker
Florida State University
Department of Computer Science B-173
207A Love Building
Tallahassee, FL 32306-4019

(904) 644-5452
(904) 644-0059 [FAX]

baker@cs.fsu.edu [e-mail]

(Electronic mail is preferred.)

The copying and distribution of IEEE balloting drafts is accomplished by the Standards Office. To report problems with reproduction of your copy, or to request additional copies of this draft, contact:

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Ann Kaczmarek
IEEE Standards Office
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08855-1331
+1 (908) 562-3811
+1 (908) 562-1571 [FAX]

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

POSIX.20 Change History

Draft 1 (November 1992) Draft prepared for mock ballot. This was derived directly from the technical report "Realtime Extension for Portable Operating Systems: 'Thin' Ada Binding, Draft 0.6" which was prepared by T.P. Baker at the Florida State University with partial funding from the HQ U.S. Army CECOM Software Engineering Directorate and the U.S. Department of Defense's Ada Technology Insertion Program (ATIP).

This document was typeset using LaTeX and TeX.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

54 *This side is intentionally blank.*

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are not 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 bee 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 the IEEE, the Institute will initiate action to prepare appropriate responses. Since IEE 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, the IEEE and the members of its technical committees are not able to provide an instant response to interpreation 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

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

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Contents

PAGE

Contents ii

Introduction ix

Section 1: General 1
1.1 Scope 1
1.2 Normative References 2
1.3 Conformance 2
1.4 Rationale for Ada Binding Approach 2
1.4.1 Threads 3
1.4.2 Handlers 5
1.4.3 Interruptible Operations 6
1.4.4 Language Independent Specification 7
1.4.5 Thin Binding Approach 8
1.4.6 Choice of Names 8
1.4.7 Implementation-Defined Extensions 9
1.4.8 Structured Types 9
1.4.9 Integer Types 10
1.4.10 Union types 10
1.4.11 Extensions to P1003.5 11
1.4.12 Organizational Problems 11

Section 2: Terminology and General Requirements 13
2.1 Conventions 13
2.1.1 Editorial Conventions 13
2.1.2 Error Handling Conventions 14
2.2 Definitions 14
2.3 General Concepts 14
2.4 Package POSIX 16
2.4.1 Optional Facilities 16
2.4.2 System Limits 18
2.4.3 Blocking Behavior Values 20
2.4.4 Option Sets 21
2.4.5 Exceptions and Error Codes 22
2.4.6 Primitive Time Types 23

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2.4.7 Time Specifications 24
2.4.8 Time Type Conversions 25
2.5 Calendar Time Conversions 25
2.5.1 Time Conversions 26
2.6 Operations on Addresses 27
2.6.1 Addresses and Address Offsets 27
2.7 Rationale 28
2.7.1 Package Structure Issues 28
2.7.2 Blocking 29
2.7.3 Error Codes 29
2.7.4 Time Types 30
2.7.5 Treatment of Limits 31
2.7.6 Blocking Behavior 32
2.7.7 Operations on Addresses 32
2.7.8 Ada 9X Considerations 33

Section 3: Revisions to Process Primitives 35
3.1 Signals 35
3.1.1 Signal Concepts 35
3.1.2 Signal Actions 37
3.1.3 Wait for Queued Signals 40
3.1.4 Queue a Signal to a Process 40
3.2 Rationale Relating to Realtime Signals Extension 41
3.2.1 Sigwaitrt and Sigtimedwait 41
3.2.2 Signal Handler Profiles 41
3.2.3 Composability Considerations 42
3.2.4 P1003.4a Considerations 42
3.2.5 P1003.5 Considerations 42
3.2.6 Ada 9X Issues 42

Section 4: Revisions to Process Environment 45
4.8 Configurable System Variables 45
4.8.1 Get Configurable System Variables 45
4.9 Rationale 48
4.9.1 Packaging 48

Section 5: Revisions to Files and Directories 49
5.6.1 File Characteristics: Header and Data Structure 49
5.6.2 File Characteristics: Header and Data Structure 49
5.6.4 Change File Modes 50

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

5.6.7 Truncate File to A Specified Length 50
5.7 Configurable Pathname Variables 51
5.7.1 Get Configurable Pathname Variables 51
5.8 Rationale for Revisions to Files and Directories 52
5.8.1 Packaging 52

Section 6: Revisions to Input and Output Primitives 53
6.5 Control Operations on Files 53
6.5.1 Data Definitions for File Control Operations 53
6.6 File Synchronization 54
6.6.1 Synchronize a File's State 54
6.6.2 Data Synchronization 54
6.7 Asynchronous Input and Output 55
6.7.0 Data Definitions for Asynchronous Input and Output 55
6.7.1 Asynchronous Read 57
6.7.2 Asynchronous Write 58
6.7.3 List Directed I/O 58
6.7.4 Retrieve Error Status of Asynchronous I/O Operation 59
6.7.5 Retrieve Return Status of Asynchronous I/O Operation 60
6.7.6 Cancel Asynchronous I/O Request 60
6.7.7 Wait for Asynchronous I/O Request 61
6.7.8 Asynchronous File Synchronization 63
6.8 Rationale for Revisions to Input and Output Primitives 63
6.8.1 Packaging 63
6.8.2 IO Control Blocks 63
6.8.3 Return Status and Error Status 64
6.8.4 File Status Flags 64
6.8.5 Asynchronous I/O Priority 64
6.8.6 Arrays 64
6.8.7 P1003.4 Questions 65
6.8.8 Ada 9X Considerations 65

Section 11: Synchronization 67
11.1 Semaphore Characteristics 67
11.1.1 Semaphore Descriptor Type 67
11.2 Semaphore Functions 68
11.2.1 Initialize an Unnamed Semaphore 68
11.2.2 Destroy an Unnamed Semaphore 68
11.2.3 Initialize/Open a Named Semaphore 69

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

11.2.4 Close a Named Semaphore 69
11.2.5 Remove a Named Semaphore 70
11.2.6 Lock a Semaphore 70
11.2.7 Unlock a Semaphore 71
11.2.8 Get the Value of a Semaphore 72
11.3 Rationale 72
11.3.1 Semaphore Creation 72
11.3.2 Semaphore Locking 72
11.3.3 Semaphore Values 73
11.3.4 P1003.4a Considerations 73

Section 12: Memory Management 75
12.1 Memory Locking Functions 75
12.1.1 Lock/Unlock a Process's Address Space 75
12.1.2 Lock/Unlock a Range of Process Address Space 76
12.2 Memory Mapping Functions 77
12.2.1 Map Process Addresses to a Memory Object 77
12.2.2 Memory Unmapping 78
12.2.3 Change Memory Protection 79
12.2.4 Memory Object Synchronization 80
12.3 Shared Memory Functions 81
12.3.1 Open a Shared Memory Object 81
12.3.2 Remove a Shared Memory Object 81
12.4 Rationale for Memory Management 82
12.4.1 Rationale for Process Memory Locking 82
12.4.2 Range Locking 83
12.4.3 Problems with Memory Mapping 84
12.4.4 `Null_Address` 85
12.4.5 Higher Level Interfaces 85
12.4.6 Shared Memory Objects 90
12.4.7 P1003.4 Questions 90
12.4.8 Ada 9X Considerations 90

Section 13: Execution Scheduling 91
13.1 Scheduling Parameters 91
13.1.1 Scheduling Parameters 91
13.2 Scheduling Policies 92
13.2.1 Scheduling Policies 92
13.3 Process Scheduling Functions 93

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

13.3.1 Set Scheduling Parameters 93
13.3.2 Get Scheduling Parameters 93
13.3.3 Set Scheduling Policy and Scheduling Parameters 94
13.3.4 Get Scheduling Policy 95
13.3.5 Yield CPU 95
13.3.6 Get Scheduling Parameter Limits 96
13.4 Rationale for Execution Scheduling 96
13.4.1 Packaging 96
13.4.2 Relationship to Ada Priorities 97
13.4.3 P1003.4a Considerations 97
13.4.4 Ada 9X Considerations 97

Section 14: Clocks and Timers 99
14.1 Data Definitions for Clocks and Timers 99
14.1.1 Time Value Specification Structures 99
14.1.2 Type Definitions 99
14.1.3 Manifest Constants 100
14.2 Clock and Timer Functions 100
14.2.1 Clocks 100
14.2.2 Create a Per-Process Timer 101
14.2.3 Delete a Per-Process Timer 102
14.2.4 Per-Process Timers 103
14.2.5 High Resolution Sleep 103
14.3 Rationale for Timers and Clocks 104
14.3.1 `Itimerspec` 105
14.3.2 Read Clock and Set Clock 106
14.3.3 Set Clock 106
14.3.4 High Resolution Sleep 106
14.3.5 Notification of Overruns 107
14.3.6 P1003.4a Considerations 108
14.3.7 Ada 9X Considerations 108

Section 15: Interprocess Message Passing 109
15.1 Data Definitions for Message Queues 109
15.1.1 Data Structures 109
15.2 Interprocess Message Passing Functions 111
15.2.1 Open a Message Queue 111
15.2.2 Close a Message Queue 111
15.2.3 Remove a Message Queue 113

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

15.2.4 Send a Message to a Message Queue . . . 113
15.2.5 Receive a Message from a Message Queue . . . 114
15.2.6 Notify Process that a Message is Available on a Queue . . . 116
15.2.7 Set Message Queue Attributes . . . 116
15.2.8 Get Message Queue Attributes . . . 117
15.3 Rationale for Interprocess Message Passing . . . 117
15.3.1 Message Queue Option Set Type . . . 117
15.3.2 Message Queue Opening Not Interruptible . . . 117
15.3.3 Priorities . . . 118
15.3.4 Typing of Messages . . . 118

Section 16: Realtime Files . . . 119
16.2 Data Definitions for Realtime Files . . . 123
16.2.1 Realtime Files Specification Structures . . . 123
16.3 Realtime File Functions . . . 130
16.3.1 Create a Realtime File . . . 130
16.3.2 Get Attributes of Realtime File . . . 131
16.3.3 Set Attributes of Realtime File . . . 131
16.3.4 Get Allocation Capabilities . . . 132
16.3.5 Get Cache Capabilities . . . 133
16.3.6 Get Buffered I/O Capabilities . . . 133
16.3.7 Get Atomic I/O Capabilities . . . 134
16.3.8 Get Direct I/O Capabilities . . . 134
16.3.9 Get Increment Lists . . . 135
16.4 Rationale for Realtime Files . . . 135
16.4.1 Buffer Allocation . . . 136
16.4.2 Constant Names . . . 136

Section A: Bibliography . . . 137

Alphabetic Topical Index . . . 139

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

This side is intentionally blank.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Introduction

(This Introduction is not a normative part of P1003.20 Information Technology — POSIX Ada Language Interfaces — Part 2: Binding for Realtime Extensions, but is included for information only.)

This document is not self-contained. For the specification of the semantics of the Ada packages specified in this report, it is necessary to refer to P1003.4, P1003.4a, and P1003.5.

Organization of This Standard

The organization of this report mostly parallels that of Draft 13 of the P1003.4 document. The present section consists of introductory material, including discussion of several global issues behind the design of the Ada-language binding. After that, the organization generally parallels that of P1003.4. The order of presentation within subsections generally consists of the code of an Ada package specification, followed by text and tables explaining the correspondence of Ada names to P1003.4 C-Language Specification names. At the end of each major section, there is a subsection entitled "Rationale"; the contents of these sections are not normative.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

This side is intentionally blank.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Information Technology — POSIX Ada Language Interfaces — Part 2: Ada Language Binding for Realtime Extensions

Section 1: General

1.1 Scope

This standard defines an extension to IEEE Standard 1003.5 Information Technology – POSIX Ada Language Interfaces – Part 1: Binding for System API, hereinafter referred to as "P1003.5", to provide an Ada binding for the interfaces specified in the IEEE draft standard Realtime Extension for Portable Operating Systems[5] (referred to in this standard as "P1003.4"), and the IEEE draft standard Thread Extension for Portable Operating Systems[1] (referred to in this standard as "P1003.4a"), to support application portability at the source code level. It is intended to be used by both application developers and system developers.

The focus of this standard is on the Ada interface to the P1003.4 draft standard. Future extensions to this may include an Ada interface for the P1003.4a draft standard. These draft standards are defined using the C programming language. Language independent definitions are supposed to be under development, but at this time insufficient progress has been made to provide a starting point for this document.

The P1003.1 standard, on which P1003.4 is based, exists presently only as a C-language binding, but working groups have been authorized to produce an Ada-language binding (hereafter referred to as P1003.5) and a language-independent specification (LIS). As of November 1992, P1003.5 has been approved as an IEEE standard, but has not yet appeared in print. The first balloting period for the P1003.1 LIS has recently closed.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Like P1003.1, P1003.4 and P1003.4a exist only as C language interfaces. These extensions to P1003.1 are now going through the IEEE ballot process. Some work was started on an LIS, but this has not kept up with changes to the base document.

1.2 Normative References

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

{1} ISO 8652: 1987[1], *Programming Languages – Ada (Endorsement of ANSI Standard 1815A-1983).* This is referred to as Ada Rm {1}throughout the rest of this document.

{2} ISO/IEC 9945-1:1990, *Information Technology – Portable Operating System Interface (POSIX) – Part 1: System Application Program Interface (API) [C Language].* This is referred to as P1003.1 throughout the rest of this document.

{3} IEEE Standard 1003.5, *Information Technology – POSIX Ada Language Interfaces – Part 1: Binding for System API.* This is referred to as P1003.5 throughout the rest of this document.

{4} IEEE Draft Standard 1003.4, *Threads Extension for Portable Operating Systems, Draft 6* (February, 1992). This is referred to as P1003.4 throughout the rest of this document.

The rationale for the present draft also makes (non-normative) reference to P1003.4a[6], P1003.4 LIS[9], Draft 2 of P1003.1 C-Language Binding[2], and Draft 4.0 of the Ada 9X Mapping Specification[8].

1.3 Conformance

No one should specify or claim conformance to the interface decribed in this document. The interface described in this document is subject to revision in future drafts, due to changes in the views of the P1003.5 working group and changes to the standards and draft standards on which this draft standard is based. Should this standard be approved, it is intended that the rules for conformance specified in P1003.5 shall apply.

1.4 Rationale for Ada Binding Approach

[1] ISO documents can be obtained from the ISO office, 1, rue de Varembe, Case Postale 56, CH-1211, Geneve 20, Switzerland/Suisse.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

By not attempting to completely describe the semantics of the P1003.4 interface, this document exemplifies the so-called "thin" language binding approach. This is different from the precedent established by P1003.5, which is a "thick" binding – i.e. a completely self-contained description of the syntax and semantics of the P1003.1 Ada interface. The reasons for taking the thin binding approach here are discussed in § 1.4.5.

The rationale subsections are provided to explain some of the reasoning behind the mapping from C to Ada, including uncertainties about interpretation of the P1003.4 interface, perceived defects in the present mapping, alternatives considered, and anticipated future interactions with the P1003.4a and Ada 9X standards activities.

The issues that pertain to the whole interface are discussed in the remainder of § 1.4. Issues pertaining to specific parts of the interface are then discussed within the sections that describe those parts.

1.4.1 Threads

The Ada language provides for tasks, which are distinct concurrent threads of control within a program. For various sound reasons, the P1003.5 standard specifiesthat each Ada program execution is a single process. The P1003.1 interface does not recognize the possibility of there being multiple concurrent threads of control within a process. Likewise, the proposed P1003.4 Realtime Extension likewise does not recognize the existence of multiple threads within a process. Thus, when a task executes a P1003.1 operation it may cause blocking of the entire process, including all the tasks within it. This is a problem for Ada applications.

Within the IEEE POSIX organization, work is now under way on a standard application program interface for multithreaded processes (Pthreads). A draft Pthreads proposal is currently in ballot, under the designation P1003.4a.

P1003.4a suggests that in a multithreaded environment the P1003.1 and P1003.4 operations which are defined as suspending the calling process should suspend only the calling thread. This offers a chance for more concurrency within Ada applications, if Ada tasks are implemented as POSIX threads. Any improvement, however, will depend on specifics of the Ada and POSIX implementations. Precisely, the normative text of P1003.4a § 3.3 says "All function defined by POSIX.1 ... and ... POSIX.4 which suspend the calling process are redefined to suspend only the calling thread." However, the non-normative text of P1003.4a § 3.3.2 says: "It has been suggested that this standard should require that no kernel operation can result in suspending more than the thread directly involved in an operation. ... This was considered ... an unreasonable burden on implementations not fully supported in the kernel." In this context, there is a possibility that the first-quoted statement might be interpreted in a weak way. That is, the POSIX.1 and POSIX.4 functions which *are required to* suspend the calling process are redefined to *not require* suspending more than the calling thread. This would be the case, for example, if threads were implemented over some form of lightweight process as servers, with more threads than lightweight processes. In this case, if the number of suspended threads ever exceeds the number of lightweight processes the entire process would be suspended.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Moving from one POSIX implementation to another, an Ada POSIX application can expect to encounter (at least) the following variations:

1. The operating system does not support Pthreads. The Ada tasking system has at least two choices:
   (a) Interleave the execution of tasks within one process, under control of the Ada vendor's own tasking kernel. POSIX operations that block the process block all tasks.
   (b) Map tasks onto multiple server processes (probably still with some interleaving), coordinated by the Ada vendor's tasking kernel. POSIX operations that block a process need not block all tasks, but might do so, if all eligible server processes are blocked.
2. The operating system supports Pthreads, via a C-language interface. The Ada tasking system now has two choices:
   (a) Ignore Pthreads, and use the implementation techniques of (1a) or (1b) above, with the resultant blocking behavior. There is a new problem: utilities in C-language libraries may assume the existence of threads. An Ada application that attempts to use such a library will have to do without tasks.
   (b) Implement tasks as threads, using the Pthread primitives in the Ada runtime system to create and control the execution of Ada tasks. If done carefully, this will allow the use of C-language libraries that make use of Pthreads. The blocking behavior, however, still depends on the Pthreads implementation:
      i. The Pthreads implementation is a so-called "library" implementation, in which case it uses implementation technique (1a) or (1b) above, with the resultant blocking behavior.
      ii. The Pthreads implementation is a "kernel" implementation, in which case system calls will not block more than the calling thread.
3. The operating system directly supports Ada tasks. The C-language binding for Pthreads now has two choices:
   (a) Ignore Ada tasking facilities, and implement Pthreads by interleaving them inside a POSIX process. Interoperability with Ada is lost.
   (b) Map threads onto Ada tasks.

Given this range of possible implementations, a portable POSIX Ada application should not rely on system calls blocking only one task. It should assume that any system call that may block might have the effect of blocking other tasks/threads within the calling- process. Of course, a good implementation of P1003.5 should attempt to avoid blocking more than one task, if this is supported by the underlying operating system.

The intent of the P1003.5 working group is to provide an Ada binding for P1003.4a, as well as P1003.4. However, as of November 1992, P1003.4a is lagging behind

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

P1003.4. It appears very likely that P1003.4 will become a standard long before P1003.4a. Therefore, it is in the interests of the Ada community to establish a standard Ada binding for P1003.4 that does not depend on P1003.4a. That is the intent of this standard — to be compatible with P1003.4a, but to not depend on it.

Fortunately, P1003.4 can be very useful without P1003.4a. An Ada application can use Ada tasks for concurrency within a program, and POSIX processes for concurrency between programs. The strength of POSIX processes is that they provide the ability to integrate systems consisting of separately developed programs, under the protection of the operating system, and to synthesize applications from programs written in different languages. By providing real-time capabilities for POSIX processes, P1003.4 makes this model useful for real-time applications.

The position of the P1003.5 working group is that Ada tasks already provide the functionality of P1003.4a threads. Therefore, should P1003.4a become a standard, the Ada binding should be expressed in terms of tasks. The requirement on Ada implementations would be for safe coexistence of Ada tasks and P1003.4a threads within the same process.

The P1003.5 working group has examined the features of P1003.4a, and found that most of the functionality is already available via existing Ada language features. The few exceptions are:

— Dynamic priority control. The Ada task priority specification pragma requires a static expression. A capability of changing the priority of a task is frequently demanded for realtime applications.

— Mutexes and condition variables. The Ada rendezvous mechanism has been criticized for being heavier-weight than is needed for many of the synchronization requirements of real-time applications. Mutexes and condition variables could provide a lighter-weight alternative.

— Thread-specific data. Some Ada applications could make good use of the ability to store data specific to a particular thread, that is accessible via the identity of the thread.

These capabilities can be provided by an Ada implementation without P1003.4a. In the absence of P1003.4a, they could be included in P1003.20 as language-specific support, however these capabilities are already included in the proposed Ada 9X language revision. Therefore, especially in light of the slow progress of P1003.4a, it probably would be best to leave them to Ada 9X.

1.4.2 Handlers

Signals are a central feature of interprocess communication in POSIX. Essentially, signals are software interrupts. By analogy to hardware interrupts, in P1003.5 signal handlers be treated as task entries, bound to the signals via Ada address clauses. This Ada-language binding for P1003.4 follows the precedent established by P1003.5. This model appears to be adequate for P1003.4.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The P1003.1 and P1003.4 C-language interfaces use procedure pointers to specify handlers for signals. Because Ada 83 does not have procedure pointers, this is not an option. The nearest thing would be to use values of type `System.Address`, obtained from procedures via the `address` attribute. This may not work with some implementations. It is also unsafe, since there is no way to check that the address is that of a procedure, or that the procedure has the proper parameter profile and is in-scope.

Handling interrupts via task entries is not the only course open for an Ada interface. A safer and higher-level Ada mechanism based on procedures is possible, using generics with formal procedure parameters. This approach was considered by the P1003.5 working group, but was not adopted, based largely on the view that the Ada language standard already specifies a mechanism for mapping interrupts to entry calls.

Taking full advantage of P1003.4a may require extending the signal handling model of P1003.5. In P1003.1, signal handlers are invoked asynchronously with respect to a process's normal thread of control, so in effect they already constitute separate threads of control. P1003.5 promotes signal handlers to full equivalence with threads, by treating them as Ada tasks. This does not allow more than one task to serve as handler for the same signal, or a signal to be sent to a specific task. The P1003.4a signal model is more complex. A signal may be directed to a specific thread; different threads may receive (different occurences of) the same signal. Supporting this in an Ada interface would appear to require making a distinction between tasks and signal handlers. This issue will need to be addressed by an Ada-language binding for P1003.4a.

As of November 1992, the Ada 9X mapping proposals include a new paradigm for interrupt handling, based on "protected procedures". This is likely to permit a closer fit between Ada and the P1003.4 model of signal handling; however, the concept of per-task interrupt/signal is still not recognized.

1.4.3 Interruptible Operations

Both P1003.1 and P1003.4 include operations that can be "interrupted" by the execution of a signal (or real-time event) handler. If the handler of such a notification "returns normally", the interrupted operation is completed prematurely, and an error code is returned. P1003.5 maps error returns to an exception. Thus, a call to an interruptible operation must be enclosed by a frame with an exception handler for `POSIX_Error` and appropriate recovery code, or all unwanted signals must be masked. This belies the illusion that the Ada binding attempts to provide — that handlers are separate tasks — since normally an operation in one Ada task cannot cause an exception to be raised in another task, except when the two are in rendezvous.

The present Ada-binding for P1003.4 follows the precedent of P1003.5 in providing an extra parameter for all interruptable operations, that specifies which signals are to be masked during the operation.

Interruptible operations still impose a special requirement on Ada implementations,

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

to ensure that interruptible operations have the correct semantics. In particular, if the Ada runtime system imposes extra code between the handler procedure invoked by the underlying operating system and the handler task specified by the Ada application (such as code to enqueue the signals as entry calls), it must ensure that the application signal handler executes before the operation interrupted by a signal returns.

1.4.4 Language Independent Specification

P1003.5 was derived from the P1003.1 C-language binding. After most of that Ada-language binding was written, ISO/IEC JTC1 SC22 WG15 decided that all the POSIX base standards should be language-independent if they are to be approved as ISO/IEC standards. The language-independent specifications (LIS) should specify the semantics. Language bindings, such as for the C-language and Ada, should only specify the syntax. That is, a language binding should be a *thin* document, which does not contain a description of the semantics but instead refers to the LIS for details. (Here, a *thin* binding is distinguished from a *direct binding*, in which there must be a one-to-one mapping between entitities in the two documents. Even a thin language binding may cast the interface in higher-level abstractions that are appropriate to the language.) The LIS for P1003.1 never caught up with the Ada-language binding. Therefore, the Ada-language binding is not based on the LIS.

The story with P1003.4 started out differently, but appears to be heading toward the same conclusion. Work on an LIS for P1003.4 started well before work on an Ada binding, but has not kept up with revisions to the base document. In March 1992, the Chair of the P1003.4 working group expected it to be completed in one or two quarters, but since that time no progress has been made on the P1003.4 LIS.

Some members of the P1003.5 working group have expressed the opinion that the thin binding approach is undesirable with Ada. The main argument for this opinion is that Ada application programmers are accustomed to using the Standard as a reference manual. They will want to use a POSIX Ada binding standard in the same way. If this is not self contained, they will be handicapped. There are also other problems, of a technical nature.

When P1003.5 was designed, any conflicts between Ada language semantics or stylistic conventions and the C language binding were resolved in favor of Ada. Since the concept of an LIS had not yet surfaced, any such conflict was viewed as a language-specific aspect of P1003.1, and is now "grandfathered" as such. In November 1992, the P1003.4 LIS is lagging far enough behind that it seems the same thing may happen. The lack of an LIS for P1003.4 may hold up a thin Ada binding from going to ballot. There is also the issue of compatibility with P1003.5, which is now an IEEE standard and is a thick binding. Since P1003.4 is a revision of P1003.1, an Ada binding for P1003.4 should be a revision of P1003.5. It would seem that this would require either doing the Ada binding for P1003.4 in a thick form consistent with that of P1003.5, or rewriting P1003.5 into a thin format. Some key officers of the POSIX organization have stated that they do not believe either of these would be necessary for standardization at the IEEE level.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

In November 1992, it is unclear how this matter will be treated by the ISO/IEC. The net effect of the ISO/IEC actions so far has been to hold back progress on international standards for languages other than C. That is, C language interfaces are being allowed to progress while the requirement for Ada language standards to be think bindings to an LIS has prevent progress at the ISO/IEC level on Ada bindings.

1.4.5 Thin Binding Approach

Experience with the large effort required to produce the (thick) P1003.5 document, the difficulty of trying to update our earlier thick binding for P1003.4, and the rapid rate of growth of the POSIX standards, motivate us to support the search for a lower-overhead approach to language bindings.

In the preparation of this document, we have assumed the standard Ada binding for P1003.4 will eventually be based on an LIS. Rather than postpone finding out about problems with the thin binding requirement, we have tried to confront them head on. Since the last draft LIS for P1003.4 is inconsistent with the current P1003.4 draft, we have structured our Ada language binding in the form of a thin binding to an LIS, but based it on the C-language binding (temporarily) instead of the LIS. This has been awkward in a few spots, but we hope these awkward spots will be eliminated when a true LIS is produced for P1003.4.

We have experimented with a very simple format, derived from that of the preliminary draft P1003.1 C-language binding. Tables are used to show the relationships between corresponding elements of the P1003.4 C-Language Specification and the Ada-language binding.

1.4.6 Choice of Names

The Ada names used in this document represent an attempt to follow the style of P1003.5. The author is not very comfortable with this decision, but felt that since this interface is an extension of P1003.5 it should be consistent in style. Both P1003.5 and the P1003.1 LIS use long, self-explanatory, names. This is in contrast to the shorter and traditional, but sometimes rather peculiar, names used in the C-language interfaces. Whether longer names are better appears to be a quasi-religous issue. Certainly there is some gain in readability with longer names, but there is also some loss of readability where the long names force unnatural line breaks. There is also an increase in source code storage size, if long names are used consistently. Perhaps the most significant issue is the apparently arbitrary difference in long names between the P1003.1 LIS and P1003.5. Given that a user of the language binding will need to refer to the LIS for semantics, translating between the different naming systems is likely to be a source of confusion. The author hopes that the groups working on the LIS and the Ada-language binding for P1003.4 will be able to achieve agreement on names.

The distinction between upper and lower case is not significant in Ada identifiers. This document follows the style of P1003.5. The rules followed are: (1) all Ada identifiers begin with an upper-case letter (because they are proper nouns, so that

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

they are easily distiguished from ordinary words, so that they can be used in initial position in a sentence); (2) if there is an underscore in an identifier, the following letter is upper case; (3) if a letter appears to be used as the initial letter of another word in the formation of an abbreviation or acronym, as in `Process_ID` or `POSIX_IO`, it is in upper case. Any deviations from these rules are unintended.

1.4.7 Implementation-Defined Extensions

P1003.5 has established precedents for the latitude an implementation is permitted in making changes to a package specification that is part of the standard. The only changes permitted to an implementation are:

a. adding **`with`** clauses, pragmas, representation specifications, and comments;

b. replacing "*implementation-defined*" by appropriate static value(s) or ranges;

c. adding or changing the packages' private parts;

d. any other changes that are lexically transparent to Ada compilers.

The intention behind these restrictions is to permit "with-clause portability". That is, it should be possible to determine whether an application relies on implementation-defined extensions by examining the with-clauses of the application. If no non-standard package names appear there, the application should be strictly conforming.

Of course, these restrictions cannot completely guarantee portability. The P1003.1 and P1003.4 standards do leave some semantic details unspecified. An application that depends on implementation semantics can be non-portable without using any non-standard names.

1.4.8 Structured Types

The C-language interfaces and draft LIS documents make use of several structured types, which would appear to map naturally to Ada record types. However, exposing record structures in a POSIX Ada interface is generally not a good idea, since the Ada interface is likely to be implemented on top of a system that is implemented in the C language. The representations of Ada records will in general be different from those of similar-looking C structures. One cause of differences is extra data (sometimes called "dope") that may be added by the Ada compiler. Other differences may derive from differences in component representations and alignment rules. C-language types that are likely to be represented differently in Ada include unsigned integers, Booleans, strings, and pointers to strings. (Union types, untyped pointers, and pointers to functions are also a problem, though such types are also problematic outside of structures.) Thus, if any record types are visible in the Ada interface it is likely that values passed as parameters to system calls will need to be transformed back and forth between Ada and C-language representations for each call. This can be a source of inefficiency.

Another issue is that the C-language interfaces and draft LIS documents do not fully specify the components of structured types; they only specify that "at least"

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

certain named components must exist and have specified types. Following this rule in an Ada interface with visible record types would mean that the use of aggregate expressions for those types would not be portable. This restriction removes one of the main potential advantages of making the record structure visible.

One way of eliminating record types from the interface is by splitting up the information which was passed as a single structure in the C-language interface, into several parameters of more primitive types. The implementation of the interface operation can then write these component values to a C-language structure, and read them back out later, if necessary. This is especially practical when the C-language structure has only a few components, or the structure may be large but the operation only uses a few of the components. However, this makes it difficult for the Ada binding to handle future extensions to the POSIX standards, which may specify additional components of the structured types. This trend has already shown up in the evolution of P1003.4 from P1003.1.

A more general approach is to treat the types that are structures in the C-language binding as private types in the Ada binding. This permits data to be represented according to the conventions of the underlying system, which may be that of any language. Operations for reading and updating components can be provided via subprograms. This solution seems to be most sensible when the record structure is complex. It is also easiest to extend. For each new logical component of the type, a new "set" procedure and "get" function are provided. As can be seen, sometimes this can get unwieldy.

1.4.9 Integer Types

The P1003.4 interface includes quite a few operations with integer valued parameters, which typically are sizes and lengths. In P1003.4 these are all of type int and long. In Ada, the only predefined type that is required to be supported is `Integer`, and the range of `Integer` is implementation dependent. The P1003.5 approach is to define a new integer type for each potentially different usage, and let the ranges be implementation-defined. However, there are a few exceptions to this rule. A notable exception is for the implementation limits. Defining a new type for every limit would have resulted in a great many new integer types, each with only one or two uses. This would have cluttered the package specifications, and required confusing type conversions for users. For such situations, if it is also clear that required range will not exceed the precision of a 16-bit signed integer, the type `Integer` has been used.

1.4.10 Union types

There are some places where the P1003.4 interface uses a C union type, notably for a one-word value that can be either an integer or a pointer. The Ada language makes this difficult. It is not possible to use a variant record in this case, because Ada requires that the discriminant be represented. For example, if the parameter of an underlying system operation only provides enough bits for one pointer or one integer to be transmitted with a real-time signal, it will not be possible for the Ada interface to transmit a pointer or integer plus a disciminant. Moreover there is an even deeper problem, since the Ada 83 type rules do not allow conversion between

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

different access types.

Where a P1003.4 interface subprogram has a parameter of a union type, we have provided a private type which the user must explicitly convert, using an instantiation of `UNCHECKED_CONVERSION`. We considered making the subprogram generic. This would make the interface a bit less cumbersome for the user, but it was considered unsafe, since it would be possible to instantiate the subprogram generic for a type whose size does not match that of the underyling data. The user of explicit unchecked conversion makes this risk more obvious to the user.

1.4.11 Extensions to P1003.5

Besides introducing new operations, P1003.4 extends the semantics of some operations defined in P1003.1. For example, the *open* operation is extended to provide two new control modes (O_SYNC and O_DSYNC). According to the IEEE TCOS-SSC draft Document Structure Plan of January 1992, there will be only one POSIX standard (LIS) document. Additions and changes such as these will be merged into the appropriate sections of the base P1003.1 document.

If this principle is applied to P1003.5, it means adding to the specifications of the packages defined there. This has been assumed in this draft of P1003.20.

1.4.12 Organizational Problems

The TCOS-SSC draft Document Structure Plan also calls for each language binding to parallel the structure of the LIS, down to the last paragraph. It is not clear whether it is practical to impose such a requirement on language bindings. This standard reflects an attempt to parallel the structure of the P1003.4 document. This has not been entirely successful. The worst example is probably § 2, whose internal organization does not parallel that of P1003.4 at all. Part of the difficulty stems from the need for additional types and operations in the Ada interface, and part of it stems from the additional structure imposed by the Ada packages. In particular, there does not seem to be any place to put the Ada package specifications. Another problem is that P1003.5 has a different organization than P1003.1. This standard is an extension to P1003.5. Since P1003.4 is structured after P1003.1, paralleling the structure of P1003.4 means deviating from the structure of P1003.5.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

425 *This side is intentionally blank.*

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Section 2: Terminology and General Requirements

2.1 Conventions

2.1.1 Editorial Conventions

Because this is a revision of P1003.5, to provide a binding for P1003.4, an attempt has been made to use section numbers here that correspond to the section numbering of P1003.4 and P1003.5. As a consequence, the section numbering of this document has intentional gaps.

The gaps in subsections correspond to sections of P1003.5 to which no changes are made. The gaps in the number of major sections corresponds to section numbers that have been reserved in the POSIX master document structure plan.

This document follows the typographical conventions shown in Table 2.1.

Table 2.1 – Typographical Conventions

Object	Typography
Ada code or identifier	`Subprogram_Name`
Ada reserved word	**`reserved_word`**
P1003.4 function	*function_name*
P1003.4 **in** argument	*argument_name*
P1003.4 **out** argument	*argument_name*
P1003.4 **in-out** argument	*argument_name*
P1003.4 datatype	*type_name*
P1003.4 datatype component	*component_name*
P1003.4 constant	CONSTANT_NAME
P1003.4 limit	{LIMIT_NAME}
P1003.4 error status value	[ERROR_STATUS_VALUE_NAME]

Correspondences between C-language names defined in P1003.4 and Ada-language names defined in this standard are shown by tables. For example, the following kind of table is used to show the correspondence between a C-language function and and Ada-language function or procedure. The indented lines specify correspondence of parameters and function return values.

Similarly, another kind of table is used to show the correspondence between members of a C-language structured type and operations which get and set components of a corresponding Ada-language private or limited private type.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2.2 – Operator Correspondence Table

C-Language		Ada-Language		Notes
C_function_name		Ada_Procedure_of_Function_Name		
C_in_parameter_name	*(I)*	Ada_In_Parameter_Name	*(I)*	
C_out_parameter_name	*(O)*	Ada_Out_Parameter_Name	*(O)*	
C_in_out_parameter_name	*(I,O)*	Ada_In_Out_Parameter_Name	*(I,O)*	
<*return-value*>		<*return-value*>		(1)

NOTES:

(1) This indicates a function return value.

Table 2.3 – Structure Type Correspondence Table

C-Language Member	Ada-Language	
	Get Value	Set Value
component_name_a	Get_A	Set_A
component_name_b	Get_B	Set_B

2.1.2 Error Handling Conventions

This standard follows the error reporting conventions established by P1003.5. The convention is that where the P1003.4 C-Language Specification specifies a function return value of -1 and a value for *errno*, the Ada-language interface specifies the exception `POSIX.POSIX_Error` and an error code that corresponds to the *errno* value. The error code associated with any instance of `POSIX_Error` can be obtained by a task via a call to `POSIX.Get_Error_Code` within a handler for that instance of the exception.

This convention, together with the content of the P1003.4 C-Language Specification, are sufficient to completely specify the error handling for each operation of this interface, outside of a small number of special cases. In what follows, the description of error handling is limited to those special cases.

The Ada-language names for constants of the `POSIX.Error_Code` type defined in P1003.5 correspond to constants in the P1003.4 C-Language Specification as specified in Table 2.4.

2.2 Definitions

The definitions of P1003.5 and P1003.4 shall apply.

2.3 General Concepts

The general concepts of P1003.5 and P1003.4 shall apply.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2.4 – P1003.5 Error Codes

C-Language	Ada-Language
[E2BIG]	Argument_List_Too_Long
[EACCES]	Permission_Denied
[EAGAIN]	Resource_Temporarily_Unavailable
[EBADF]	Bad_File_Descriptor
[EBUSY]	Resource_Busy
[ECHILD]	No_Child_Process
[EDEADLK]	Resource_Deadlock_Avoided
[EEXIST]	File_Exists
[EFBIG]	File_Too_Large
[EINTR]	Interrupted_Operation
[EINVAL]	Invalid_Argument
[EIO]	Input_Output_Error
[EISDIR]	Is_A_Directory
[EMFILE]	Too_Many_Open_Files
[EMLINK]	Too_Many_Links
[ENAMETOOLONG]	Filename_Too_Long
[ENFILE]	Too_Many_Open_Files_In_System
[ENODEV]	No_Such_Operation_On_Device
[ENOENT]	No_Such_File_Or_Directory
[ENOEXEC]	Exec_Format_Error
[ENOLCK]	No_Locks_Available
[ENOMEM]	Not_Enough_Space
[ENOSPC]	No_Space_Left_On_Device
[ENOSYS]	Operation_Not_Implemented
[ENOTDIR]	Not_A_Directory
[ENOTEMPTY]	Directory_Not_Empty
[ENOTTY]	Inappropriate_IO_Control_Operation
[ENXIO]	No_Such_Device_Or_Address
[EPERM]	Operation_Not_Permitted
[EPIPE]	Broken_Pipe
[EROFS]	Read_Only_File_System
[ESPIPE]	Invalid_Seek
[ESRCH]	No_Such_Process
[EXDEV]	Improper_Link

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2.4 Package POSIX

The package POSIX, defined by P1003.5 contains declarations that are used by other interface packages. This standard adds declarations to this package, for: constant declarations of error codes that may be raised by operations defined in other packages; types and operations that are used to represent time for operations defined in other packages; limits on the use of the interfaces specified by this standard imposed by the particular implementation.

```
with System;
package POSIX is
    -- Declarations carried forward from P1003.5 are not shown here.
      ...
    -- §2.4.1 Optional Facilities   ...
    -- §2.4.2 System Limits   ...
    -- §2.4.3 Blocking Behavior Values   ...
    -- §2.4.4 Option Sets   ...
    -- §2.4.5 Exceptions and Error Codes   ...
    -- §2.4.6 Primitive Time Types   ...
    -- §2.4.7 Time Specifications   ...
    -- §2.4.8 Time Type Conversions   ...
private
    -- Declarations carried forward from P1003.5 are not shown here.
      ...
     type Timespec is impl.-defined;
end POSIX;
```

2.4.1 Optional Facilities

2.4.1.1 Synopsis

```
    subtype Async_IO_Support is Boolean range impl.-defined static range;
    subtype Semaphores_Support is Boolean range impl.-defined static range;
    subtype File_Sync_Support is Boolean range impl.-defined static range;
    subtype Mapped_Files_Support is Boolean range impl.-defined static range;
    subtype Memory_Locking_Support is Boolean range impl.-defined static range;
    subtype Memory_Range_Locking_Support is Boolean range impl.-defined static range;
    subtype Memory_Protection_Support is Boolean range impl.-defined static range;
    subtype Message_Passing_Support is Boolean range impl.-defined static range;
    subtype Prioritized_IO_Support is Boolean range impl.-defined static range;
    subtype Priority_Scheduling_Support is Boolean range impl.-defined static range;
    subtype Realtime_Files_Support is Boolean range impl.-defined static range;
    subtype Realtime_Signals_Support is Boolean range impl.-defined static range;
    subtype Shared_Memory_Objects_Support is Boolean range impl.-defined static range;
    subtype Sync_IO_Support is Boolean range impl.-defined static range;
    subtype Timers_Support is Boolean range impl.-defined static range;
```

Certain features of this standard are optional. For each option, the P1003.4 C-Language Specification defines an optional compile-time constant, whose name is of the form {_POSIX_*XXX*}, and a configurable system variable, whose name is of the form {_SC_POSIX_*XXX*}.

The Ada-language interfaces corresponding to these options are divided into packages. In some cases, there are options within a package. The correspondence of the

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

options to Ada-language packages is shown in Table 2.5.

Table 2.5 – Options & Packages

Option	Ada Package	Notes
{_POSIX_ASYNCHRONOUS_IO}	`POSIX_Asynchronous_IO`	
{_POSIX_FSYNC}	`POSIX_IO`	
{_POSIX_SYNCHRONIZED_IO}	`POSIX_IO`	(1)
{_POSIX_MAPPED_FILES}	`POSIX_Memory_Mapping`	(2)
{_POSIX_MEMORY_PROTECTION}	`POSIX_Memory_Mapping`	(2)
{_POSIX_MEMLOCK}	`POSIX_Memory_Locking`	(3)
{_POSIX_MEMLOCK_RANGE}	`POSIX_Memory_Range_Locking`	(3)
{_POSIX_MESSAGE_PASSING}	`POSIX_Message_Passing`	
{_POSIX_PRIORITIZED_IO}	`POSIX_Asynchronous_IO`	
{_POSIX_PRIORITY_SCHEDULING}	`POSIX_Priority_Scheduling`	
{_POSIX_REALTIME_FILES}	`POSIX_Realtime_Files`	
{_POSIX_REALTIME_SIGNALS}	`POSIX_Signals`	
{_POSIX_SEMAPHORES}	`POSIX_Semaphores`	
{_POSIX_SHARED_MEMORY_OBJECTS}	`POSIX_Shared_Memory_Objects`	(2)
{_POSIX_TIMERS}	`POSIX_Timers`	

NOTES:

(1) The *fsync* and *fdatasync* operations are supported by procedures in the package `POSIX_IO`, but this option also affects POSIX operations defined in other package.

(2) Interfaces involving memory mapping and memory protection are in `POSIX_Memory_Mapping`. Interfaces involving shared memory object creation and destruction are in `POSIX_Shared_Memory_Objects`.

(3) If {_POSIX_MEMLOCK_RANGE} is supported then {_POSIX_MEMLOCK} shall also be supported.

The P1003.4 C-Language Specification specifies symbolic constants and configurable system variables that describe whether specific options are supported by the implementation. For each P1003.4 C-Language Specification symbolic constant.

If the range of the subtype is `True..True`, the option shall always be supported. If the range is `False..True` or `False..False` the function may not be supported[1]. If the range of the subtype is `False..True`, more information about whether the option is supported can be obtained by calling a function at run time (§ 4.8)[2].

The Ada-language subtypes for option support shall correspond to the P1003.4 C-Language Specification symbolic constants as shown in Table 2.6.

The Ada-language functions to query whether options are supported are described in § 4.8.1.

2.4.1.2 References

options, § 4.8.1, P1003.4 § 2.9.3.

[1] P1003.4 apparently allows the system to support an option, even when it replies `False` to queries whether the option is not supported.

[2] The function must always return `False` if the range is `False..False`, and always return `True` if the range is `True..True`.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2.6 – Compile-Time Ranges for Options

C-Language	Ada-Language
{_POSIX_ASYNCHRONOUS_IO}	Async_IO_Support
{_POSIX_FSYNC}	File_Sync_Support
{_POSIX_MAPPED_FILES}	Mapped_Files_Support
{_POSIX_MEMLOCK}	Memory_Locking_Support
{_POSIX_MEMLOCK_RANGE}	Memory_Range_Locking_Support
{_POSIX_MEMORY_PROTECTION}	Memory_Protection_Support
{_POSIX_MESSAGE_PASSING}	Message_Passing_Support
{_POSIX_PRIORITIZED_IO}	Prioritized_IO_Support
{_POSIX_PRIORITY_SCHEDULING}	Priority_Scheduling_Support
{_POSIX_REALTIME_FILES}	Realtime_Files_Support
{_POSIX_REALTIME_SIGNALS}	Realtime_Signals_Support
{_POSIX_SEMAPHORES}	Semaphore_Support
{_POSIX_SHARED_MEMORY_OBJECTS}	Shared_Memory_Objects_Support
{_POSIX_SYNCHRONIZED_IO}	Sync_IO_Support
{_POSIX_TIMERS}	Timers_Support

2.4.2 System Limits

2.4.2.1 Synopsis

```
subtype Page_Size_Range is Natural range 1..impl.-defined static const.;
Portable_List_IO_Max: constant Natural:= 2;
subtype List_IO_Maxima is Natural range impl.-defined static range;
Portable_Async_IO_Max: constant Natural:= 1;
subtype Async_IO_Maxima is Natural range impl.-defined static range;
Portable_Clock_Resolution_Min: constant:= 20_000_000; -- nanoseconds
Portable_Timer_Overruns_Max: constant Natural:= 32;
subtype Timer_Overruns_Maxima is Natural range impl.-defined static range;
Portable_Message_Queue_Open_Max: constant Natural:= 8;
subtype Message_Queue_Open_Maxima is Natural range impl.-defined static range;
Portable_Message_Queue_Priority_Max: constant Natural:= 32;
subtype Message_Queue_Priority_Maxima is Natural range impl.-defined static range;
Portable_Realtime_Signals_Max: constant Natural:= 8;
subtype Realtime_Signals_Maxima is Natural range impl.-defined static range;
Portable_Semaphores_Max: constant Natural:= 256;
subtype Semaphores_Maxima is Natural range impl.-defined static range;
Portable_Semaphores_Value_Max: constant Natural:= 32_767;
subtype Semaphores_Value_Maxima is Natural range impl.-defined static range;
Portable_Queued_Signals_Max: constant Natural:= 32;
subtype Queued_Signals_Maxima is Natural range impl.-defined static range;
Portable_Timers_Max: constant Natural:= 32;
subtype Timers_Maxima is Natural range impl.-defined static range;
```

2.4.2.2 Description

An implementation may impose maximum or minimum values for certain numbers. An example of a maximum is an upper bound on the number of semaphores. An example of a minimum is a lower bound on the resolution of a real-time clock.

For each maximum or minimum, the P1003.4 C-Language Specification specifies a symbolic constant for the most restrictive value on a system conforming to this

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

standard. The constant has a name of the form _POSIX_*XXX*_MAX or _POSIX_*XXX*_MIN. The P1003.4 C-Language Specification also specifies a symbolic constant for the actual value of each maximum or minimum, which is only defined if the actual limit is run-time invariant. The name of such a constant is of the form *XXX*_MAX or *XXX*_MIN. Finally, the P1003.4 C-Language Specification specifies a configurable system variable for each maximum or minimum, whose value at runtime indicates the actual limit. The name of such a configurable system variable is of the form **_SC_*XXX*_MAX** or **_SC_*XXX*_MIN**.

An implementation also specifies a memory page size. This serves as a lower bound on the granularity of memory mapping and process memory locking. As with the other limits, there is an Ada-language subtype that specifies the range of possible page sizes that may be supported by the implementation. If the page size is run-time invariant, the upper and lower bounds of the range `Page_Size_Range` shall both be equal to the actual page size. There is an Ada-language function to obtain the actual page size at run time.

The Ada-language constants for the most restrictive values permitted shall correspond to constants in the P1003.4 C-Language Specification as shown in Table 2.7.

Table 2.7 – Portable Limits

C-Language	Ada-Language
{_POSIX_AIO_LISTIO_MAX}	`Portable_List_IO_Max`
{_POSIX_AIO_MAX}	`Portable_Async_IO_Max`
{_POSIX_CLOCKRES_MIN}	`Portable_Clock_Resolution_Min`
{_POSIX_DELAYTIMER_MAX}	`Portable_Timer_Overruns_Max`
{_POSIX_MQ_OPEN_MAX}	`Portable_Message_Queue_Open_Max`
{_POSIX_MQ_PRIO_MAX}	`Portable_Message_Queue_Priority_Max`
{_POSIX_RTSIG_MAX}	`Portable_Realtime_Signals_Max`
{_POSIX_SIGQUEUE_MAX}	`Portable_Queued_Signals_Max`
{_POSIX_SEM_NSEMS_MAX}	`Portable_Semaphores_Max`
{_POSIX_SEM_VALUE_MAX}	`Portable_Semaphores_Value_Max`
{_POSIX_TIMER_MAX}	`Portable_Timers_Max`

A conforming implementation shall not impose a maximum that is less than the portable maximum or a minimum that is greater than the portable minimum. A strictly conforming application shall not require a value that is greater than the portable maximum, or less than the portable minimum.

Ada-language subtypes for run-time invariant limits shall correspond to the constants of the P1003.4 C-Language Specification as shown in Table 2.8.

Note that the P1003.4 C-Language Specification does not define a runtime invariant limit for {INTERVAL_MAX}.

Each of these subtypes defines a static range of values that includes the actual limit imposed by the implementation. If the limit is a maximum, *XXX*`_Maxima'FIRST` must be greater than or equal to the portable maximum. If the limit is a minimum, *XXX*`_Minima'LAST` must be less than or equal to the portable minimum. If the actual limit is run-time invariant, the upper and lower bounds of this range shall be the same. If the actual limit is indeterminate, the range of the subtype should

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2.8 – Run-Time Invariant Limits

C-Language	Ada-Language
{PAGESIZE}	Page_Size_Range
{AIO_LISTIO_MAX}	List_IO_Maxima
{AIO_MAX}	Async_IO_Maxima
{AIO_PRIO_DELTA_MAX}	Async_IO_Priority_Delta_Maxima
{DELAYTIMER_MAX}	Timer_Overruns_Maxima
{MQ_OPEN_MAX}	Message_Queue_Open_Maxima
{MQ_PRIO_MAX}	Message_Queue_Priority_Maxima
{RTSIG_MAX}	Realtime_Signals_Maxima
{SIGQUEUE_MAX}	Queued_Signals_Maxima
{SEM_NSEMS_MAX}	Semaphores_Maxima
{SEM_VALUE_MAX}	Semaphores_Value_Maxima
{TIMER_MAX}	Timers_Maxima

include the full range of values that the actual limit may take on.

The Ada-language functions returning the actual run-time limits are specified in §4.8.1.

The values reported to an Ada-language application by the constants, ranges, and functions defined in this standard shall reflect the actual limits as they apply to the Ada application. For example, if the runtime system of the Ada language implementation uses any semaphores or per-process timers, the values reported by `Semaphores_Max` and `Timers_Max` to an Ada application shall be reduced from that supported by the underlying operating system to reflect the semaphores and timers already used by the Ada runtime system.

In particular, the constants defining portable maxima and minima shall accurately reflect the resources available to the application. As a consequence of the limitations on implementation modifications to the POSIX package specifications (see P1003.5), an implementation shall not modify the values of any of these constants.

In some cases this may imply that the Ada language implementation cannot use a given feature for its own purposes. For example, if the underlying implementation only supports `Portable_Semaphores_Max` semaphores the Ada runtime system cannot use any semaphores for its own purposes, since otherwise the full number of semaphores available to the application would be reduced below the portable limit.

2.4.2.3 References

{PAGESIZE}, P1003.4 §2.8.4; {_SC_PAGESIZE}, P1003.4 §4.8.1; minima, P1003.4 §2.8.2; maxima, P1003.4 §2.8.7; compile-time limits, P1003.4 §2.9.3; system configurable limits, P1003.4 §4.8.1.

2.4.3 Blocking Behavior Values

2.4.3.1 Synopsis

```
   subtype Async_IO_Blocking_Behavior is POSIX.Blocking_Behavior
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
      range impl.-defined static range;
   subtype Semaphore_Blocking_Behavior is POSIX.Blocking_Behavior
      range impl.-defined static range;
   subtype Message_Queue_Blocking_Behavior is POSIX.Blocking_Behavior
      range impl.-defined static range;
   subtype Shared_Memory_Blocking_Behavior is POSIX.Blocking_Behavior
      range impl.-defined static range;
   subtype Realtime_File_Create_Blocking_Behavior is POSIX.Blocking_Behavior
      range impl.-defined static range;
   subtype Realtime_Signal_Blocking_Behavior is POSIX.Blocking_Behavior
      range impl.-defined static range;
   subtype Timer_Blocking_Behavior is POSIX.Blocking_Behavior
      range impl.-defined static range;
```

2.4.3.2 Description

Language-specific subtypes provide information about whether blocking is per-task or per-process. The name of each subtype corresponds to the name of a package. If the range is `Tasks..Tasks`, all the blocking operations defined in the corresponding package shall block only the calling task. If the range is `Program..Program`, all the blocking operations defined in the corresponding package shall block the entire process. Otherwise, the blocking operations defined in the package shall block at least the calling task, possibly some other tasks, and possibly the entire process. The correspondence of constants to packages shall be as shown in Table 2.9.

Table 2.9 – Blocking Behavior Subtypes

Behavior	Package
Async_IO_Blocking_Behavior	POSIX_Asynchronous_IO
Semaphore_Blocking_Behavior	POSIX_Semaphores
Message_Queue_Blocking_Behavior	POSIX_Message_Queues
Shared_Memory_Blocking_Behavior	POSIX_Shared_Memory_Objects
Realtime_File_Blocking_Behavior	POSIX_Realtime_Files
Realtime_Signal_Blocking_Behavior	POSIX_Signals
Timer_Blocking_Behavior	POSIX_Timers

2.4.3.3 References

`Blocking_Behavior`, P1003.5 §2.

2.4.4 Option Sets

2.4.4.1 Synopsis

```
   function "<=" (L,R: POSIX.Option_Set) return Boolean;
   function "<"  (L,R: POSIX.Option_Set) return Boolean;
   function ">=" (L,R: POSIX.Option_Set) return Boolean;
   function ">"  (L,R: POSIX.Option_Set) return Boolean;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2.4.4.2 Description

The type POSIX.Option_Set, defined in P1003.5, provides an abstraction for sets of options, from which types are derived to specify options for certain operations.

```
type Option_Set is private;
-- "=" and ":=" have value-semantics
function Empty_Set return Option_Set;
function "+"(L,R: Option_Set) return Option_Set;
function "-"(L,R: Option_Set) return Option_Set;
```

Option set variables are implicitly initialized the the empty value. The "+" operation returns the union of two sets. The binary "-" operation returns the members of the left argument that are not members of the right argument.

In the various interface packages, different option set types are derived from POSIX.-Option_Set. These types inherit the operations of POSIX.Option_Set. Constants of each type are defined, representing singleton options. (Since the original option set type is private, the expressions giving the values of these constants need to be specified using unchecked conversion, or an implementation defined means.)

For example, a set of memory protection options might be specified, with constants Allow_Read, Allow_Write, and Allow_Execute. These can then be combined using the operations, as in Allow_Read + Allow_Write + Allow_Execute.

Here, additional operations on the Option_Set type are specified to allow comparison of two option sets. If L and R are option sets, the operations have the meanings shown in Table 2.10.

Table 2.10 – Option Set Comparisons

Relation	Meaning
L <= R	L is a subset of R
L < R	L is a proper subset of R
L >= R	R is a subset of L
L > R	R is a proper subset of L

2.4.4.3 References

Option_Set, P1003.5 § 2; Open_Option_Set, P1003.5 § 6.

2.4.5 Exceptions and Error Codes

2.4.5.1 Synopsis

```
Operation_Canceled:      constant POSIX.Error_Code:= impl.-defined;
Bad_Message:             constant POSIX.Error_Code:= impl.-defined;
Message_Too_Long:        constant POSIX.Error_Code:= impl.-defined;
Operation_In_Progress:   constant POSIX.Error_Code:= impl.-defined;
Operation_Not_Supported: constant POSIX.Error_Code:= impl.-defined;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2.4.5.2 Description

In addition to the error code values defined by P1003.5, this standard defines certain new error codes. The correspondence of P1003.4 C-Language Specification error codes to Ada-language error codes is shown in Table 2.11.

Table 2.11 – P1003.4 Error Codes

C-Language	Ada-Language	Notes
0	`No_Error`	
[ECANCELED]	`Operation_Canceled`	
[EBADMSG]	`Bad_Message`	
[EINPROGRESS]	`Operation_In_Progress`	
[EMSGSIZE]	`Message_Too_Long`	
[ENOTSUP]	`Operation_Not_Supported`	

NOTES:

(1) The value `No_Error` is returned by `POSIX_Asynchronous_IO.Get_Error_Code` for the case that an I/O operation has completed successfully.

2.4.5.3 References

[ECANCELED], P1003.4 § 2.4; [EBADMSG], P1003.4 § 2.4; [EINPROGRESS], P1003.4 § 2.4; [EMSGSIZE], P1003.4 § 2.4; [ENOTSUP], P1003.4 § 2.4.

2.4.6 Primitive Time Types

2.4.6.1 Synopsis

```
type Seconds is range impl.-defined static range;
-- must include at least -(2**31 - 1)..(2**31 - 1)
type Nanoseconds_Base is range impl.-defined static range;
-- must include at least -(2**31 - 1)..(2**31 - 1)
subtype Nanoseconds is Nanoseconds_Base range 0..(10**9)-1;
```

2.4.6.2 Description

These types are used together to represent large durations with nanosecond precision.

Table 2.12 – Time Specification

C-Language	Ada-Language
time_t	`Seconds`
long	`Nanoseconds_Base`
long	`Nanoseconds`

`Nanoseconds` is a subtype which is interpreted as a non-negative count of nanoseconds, amounting to less than one second. `Nanoseconds_Base` is the base type of this subtype, which includes negative values and nanosecond counts amounting to more than one second. This corresponds to the *long* type used as the nanosecond component of the *timespec* structure.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2.4.6.3 References

time_t, P1003.4 § 14.1.1.

2.4.7 Time Specifications

2.4.7.1 Synopsis

```
type Timespec is private;
function Get_Seconds(T: Timespec) return Seconds;
procedure Set_Seconds
   (T:     in out Timespec;
    Value: in Seconds);
function Get_Nanoseconds(T: Timespec) return Nanoseconds;
procedure Set_Nanoseconds
   (T:     in out Timespec;
    Value: in Nanoseconds);
function "+"(L,R: Timespec) return Timespec;
function "-"(L,R: Timespec) return Timespec;
function "<"(L,R: Timespec) return Timespec;
function ">"(L,R: Timespec) return Timespec;
```

2.4.7.2 Description

The Ada type `Timespec` corresponds to the P1003.4 C-Language Specification *timespec* type. The Ada-language names for read and write operations of the `Timespec` type shall correspond to members of the *timespec* type as shown in Table 2.13.

Table 2.13 – The *timespec* Structure

C-Language Member	Ada-Language	
	Get Value	Set Value
tv_sec	`Get_Seconds`	`Set_Seconds`
tv_nsec	`Get_Nanoseconds`	`Set_Nanoseconds`

Values of type `Timespec` represent relative times; that is, the lengths of time intervals. `Timespec` is also used to specify absolute times, as an offset relative to a reference time known as "the Epoch". (The Epoch is defined in P1003.1 and P1003.5 to be the time 0 hours, 0 minutes, 0 seconds, January 1, 1970, Coordinated Universal Time.)

A value $(sec, nsec)$ of the type `Timespec` denotes a duration of $sec + nsec \cdot 10^{-9}$ seconds. That is, sec is interpreted as a signed number of whole seconds and $nsec$ as a count of nanoseconds.

2.4.7.3 References

timespec, P1003.4 § 14.1.1.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2.4.8 Time Type Conversions

Functions are provided to permit convenient conversion of times between `Standard.Duration` and `Timespec` representations, without unnecessary range limitations or loss of precision. (For conversions with `Calendar.Time` see § 2.5.)

2.4.8.1 Synopsis

```
function To_Duration(T: Timespec) return Duration;
function To_Timespec(D: Duration) return Timespec;
```

2.4.8.2 Description

`To_Duration` converts a value of the type `Timespec` to a value of the type `Duration`, if the value can be expressed in that range. Otherwise, it raises `Constraint_Error`.

`To_Timespec` converts a value of the type `Duration` to a value of the type `Timespec`.

2.4.8.3 Error Handling

`Constraint_Error` shall be raised by the conversion functions if the value of the argument is outside the range that can be converted to the return type.

These operations shall not raise an exception under any other conditions.

2.4.8.4 References

`To_Calendar_Time`, § 2.5; `To_Timespec`, § 2.5; `Split`, § 2.5; `Timespec_Of`, § 2.5.

2.5 Calendar Time Conversions

This section describes the services which this standard provides for conversion betweeen values of the `POSIX.Timespec` and time representations used by the Ada `Calendar` package. Functions are provided to permit convenient conversion of times between `Calendar.Time` and `Timespec` representations, without unnecessary range limitations or loss of precision. (For conversions with `Standard.Duration` see § 2.4.8.)

Specification of Package `POSIX_Time_Conversions`

```
with Calendar;
with POSIX;                          -- from P1003.5
package POSIX_Time_Conversions is
   -- § 2.5.1 Time Conversions ...
end POSIX_Time_Conversions;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2.5.1 Time Conversions

2.5.1.1 Synopsis

```
function To_Calendar_Time
  (D: POSIX.Timespec) return Calendar.Time;
function To_Timespec
  (T: Calendar.Time) return POSIX.Timespec;
procedure Split
    (Date:    in  POSIX.Timespec;
     Year:    out Calendar.Year_Number;
     Month:   out Calendar.Month_Number;
     Day:     out Calendar.Day_Number;
     Seconds: out Calendar.Day_Duration);
function Timespec_Of
    (Year:    Calendar.Year_Number;
     Month:   Calendar.Month_Number;
     Day:     Calendar.Day_Number;
     Seconds: Calendar.Day_Duration := 0.0)
  return POSIX.Timespec;
```

2.5.1.2 Description

`To_Calendar_Time` converts a value of the type `Timespec`, interpreted as a duration since the Epoch to a value of the type `Calendar.Time`

`To_Timespec` converts a value of the type `Calendar.Time` to a value of the type `Timespec`. When converting a value of the type `Calendar.Time`, the value returned shall be interpreted as a duration since the Epoch.

`Split` interprets a value of the type `Timespec` as a date specified by an offset from the Epoch.

`Timespec_Of` returns the value of the type `Timespec` specifying a date as an offset relative to the Epoch.

2.5.1.3 Error Handling

`Constraint_Error` shall be raised by the conversion functions if the value of the argument is outside the range that can be converted to the return type.

These operations shall not raise an exception under any other conditions.

2.5.1.4 References

`To_Duration` § 2.4.8; `To_Timespec` § 2.4.8.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2.6 Operations on Addresses

2.6.1 Addresses and Address Offsets

2.6.1.1 Synopsis

2.6.1.2 Description

The type `System.Address` shall not be a limited private type. `System.Address` may be a private type (recommended), an integer type, or a composite type (e.g. record type). Assignment, equality and inequality shall be supported on `System.Address`.

The value of `Null_Address` shall correspond to the "*addr* value of zero" for *mmap* in P1003.4 § 12.2 (p 116, line 277).

The type `Address_Offset` shall be declared as an integer type (either signed or unsigned), with an implementation-defined range including the value 0. An implementation may support negative values in `Address_Offset`, but is not required to do so.

The `Address_Offset` value zero is the identity value for address arithmetic. That is, the following relationships must hold for `"+"`, `"-"` and `"mod"` for all addresses `X`:

— `X = X + Address_Offset'(0)`

— `X = Address_Offset'(0) + X`

— `X = X - Address_Offset'(0)`

The following relationship must hold for the `Image` and `Value` functions, for all addresses `X`:

— `X = Value(Image(X))`

The format of the string returned by `Image` and the string accepted by the `Value` is implementation defined.

This standard also specifies the recommendations of UI-0043, regarding the meaning of the `address` attribute. That is:

— Address clauses and address attributes designate the address of the first component of user (not compiler-generated) data.

— The address specified by the `address` attribute of an object should be the address provided by the address clause for the object.

— Given a record type with a record representation clause, `address` of a record object should be the address of the component at storage place 0, as specified in the record representation.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

These operations are provided to simplify the computation of page-aligned specifications for memory range locking (§ 12.1.2) and memory mapping (§ 12.2).

The values computed by these functions are specified as follows, where `Page_Size` denotes `POSIX_Configurable_System_Limits.Page_Size`:

— `Align_To_Page(A) = A - (A mod Page_Size)`

— `Align_To_Page(A, L) = L + (A mod Page_Size)`

— `Length(B) = B/System.Storage_Unit`

That is:

— If `A` is an address (file offset) `Align_To_Page(A)` shall be the address (file offset) of the first page that includes the storage cell with address (offset) `A`.

— The range of cells with addresses (offsets) `A..A+L` shall be included in the range `Align_To_Page(A) .. Align_To_Page(A) + Align_To_Page(A,L)`.

— `Length(X'address)` shall be the number of addressable storage cells occupied by an object `X`.

— `X'address` shall be the address of the first storage cell occupied by an object `X`.

2.6.1.3 Error Handling

Any of these operations returning an address or address offset is permitted to raise `Address_Error` if no reasonable interpretation of the requested operation exists on the machine.

The other operations are permitted to raise `Constraint_Error` if the result value would be outside the range of the result type.

2.6.1.4 References

`Lock_Range`, `Unlock_Range`, § 12.1.2; `Map_Memory`, `Unmap_Memory`, § 12.2.

2.7 Rationale

2.7.1 Package Structure Issues

The decision of how to group declarations into packages was difficult in some cases, due to conflicts between the organizations of the P1003.5 packages, the organization of the P1003.4 document, and the recent direction of the IEEE global document structure plan.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

This standard extends the package `POSIX` of P1003.5. Specifically, it defines additional error codes, additional limits, and certain additional types of global interest. As mentioned in § 1.4.11, the IEEE TCDOS-SSC draft Document Structure Plan of January 1992, calls for extensions to the base standard will be inserted into the section of the standard which they extend. Therefore, the material here should go into Section 2 of the P1003.5 document.

2.7.2 Blocking

The treatment of blocking behavior here may need reconsideration. It may be better to specify one global value for the behavior of potentially blocking POSIX operations with respect to Ada tasks, since it is generally unlikely that this behavior would be different for most operations.

Ideally, blocking should always be per-task. It is just that this is impractical to achieve for some system calls on some operating systems, so that to require it for all implementations of the Ada binding would prevent use of the Ada binding on these systems. This situation is analogous to the interpretation of the Ada language standard[11] which allows an implementation to omit support for features which it is impractical to implement on a particular execution platform. Since POSIX does not have such a rule, the best approximation seems to be to explicitly require that blocking behavior be `Tasks`, unless this cannot be implemented using the underlying system. An implementor would be required to document the reasons for any cases in which the blocking behavior is otherwise. The intent here is again similar to the Ada validation policy, which requires the Ada language implementor to justify any claim that a feature cannot be implemented practically on a particular machine.

It has been argued that since the scope of this draft does not include P1003.4a, no such requirement should be included. Others have argued that this requirement is already implicit in the Ada language task semantics, so there is no need to wait for P1003.4a to include this requirement.

2.7.3 Error Codes

P1003.4 extends the set of error code values defined in P1003.1. Likewise, this document extends the set of error codes defined in P1003.5.

The list of error codes here do not all exactly correspond to those in P1003.4 § 2.4. `No_Error` here corresponds to the P1003.4 C-Language Specification error code zero, which is used in Asynchronous I/O when there has been no error.

Perhaps one of the existing error codes should be eliminated. [ENOTSUP] is apparently only used for *mmap*,*munmap*, in the case certain functionality is not supported by the implementation. This is a case of non-uniform treatment of such restrictions in P1003.4. Maybe the P1003.4 working group should consider using one of the other error codes here, such as [ENOSYS].

Certain error codes are apparently never returned via *errno*. They are only associated with an asynchronous I/O control block, to indicate the status of an I/O operation.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The *aio_error* function extracts these values using the I/O control block as "handle". Presumably, the reason for these being of the same type as other error codes is that the I/O operations may also run into errors, which need to be returned via the same interface.

[ECANCELED] means that an asynchronous operation was canceled before completion.

2.7.4 Time Types

The `Timespec` type is in the package `POSIX`, rather than in the `POSIX_Timers` package, because it is used as the type of a parameter of `POSIX_Signals.Await_Signal_Or_Timeout`. Since `POSIX_Timers` depends on `POSIX_Signals`, putting `Timespec` in `POSIX_Timers` would create a circular compilation dependency.

The `Nanoseconds` component of a timespec is required to be a count in the range zero to one second. This constraint is expressed by the `Nanoseconds` subtype. The `Nanoseconds_Base` type is provided for situations where arithmetic must be performed on nanosecond counts and the intermediate results may need to be outside the range of `Nanoseconds`.

The conversion functions for `Calendar.Time` are separate compilation units, so that `POSIX_Timers` does not depend on `Calendar`. This allows `Calendar` to be implemented using `POSIX_Timers`.

The idea of providing separate types for duration and time, as the Ada language does, was considered. The decision to use a single type is based on simplicity and consistency with the P1003.4 interface.

The possibility of making `Timespec` a private type was considered. Certainly, P1003.4 specifies only that the *timespec* structure "includes at least" two members. However, the semantics of this type make it unlikely that it would have other visible components, and there appears to be no reason the Ada implementation could not make the representation identical to that of the C-language structure. Given this, the advantage to the programmer of being able to use aggregate notation outweighed other considerations.

Another alternative considered is to use the standard `Duration` and `Calendar.Time` types directly. This would permit elimination of the separate type `Timespec`, but would mean requiring that the implementation of these provide at least as much range and precision as the *timespec* type does. The decision not to do this, is based on the argument that it would impose too much overhead on non-real-time applications, In particular, if `Duration` is represented as a 32-bit signed integer, it is not possible to satisfy both the P1003.4 requirement for nanosecond precision and the Ada requirement for a range of plus or minus one day. On the other hand, if `Duration` and `Time` are both represented as a 64-bit integer, there would be extra overhead for conversion between the Ada representation and the C time structure. The representation defined by the P1003.4 C language binding (which is a structure containing second and nanosecond counts) is un-natural for an Ada fixed point type. It is therefore likely that an Ada language interface would have to convert arguments

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

between representations (in both directions). Moreover, supporting this type would imply supporting general 64-bit fixed-point arithmetic.

The `Nanoseconds` type was not made a fixed-point (real) type, for two reasons. First, some Ada compilers do not permit exact specification of decimal `small` values for fixed-point types. Lacking this, there might be error introduced in converting between true nanoseconds the fixed-point type. Second, the Ada semantics for fixed-point types would require the implementation to support multiplication and division operations with greater precision than is required for time calculations, resulting in unnecessary overhead for the implementation.

The constant `Portable_Clock_Resolution_Min` is untyped, rather than having type `Nanoseconds_Base`, since this would force the declaration of the time types to precede the portable constants.

2.7.5 Treatment of Limits

The treatment of limits is modeled after that in P1003.5. However, the resulting proliferation of subprograms suggests that perhaps a single operation (such as the C-language *sysconfig* interface) would have been a more economical approach for P1003.5.

Though it is not specified explicitly here, this interface inherits from P1003.5 §1.3.1.3 the requirement that an implementation provide declarations for all the interfaces defined here. For options that are not supported, the implementation raises `POSIX_Error` if the application attempts to call any of the subprograms. This rule of P1003.5 may deserve reconsideration, now that there are so many more options. It may be reasonable to allow an implemention to simply leave out of its library the package specifications for options that are not supported by that implementation.

Note that it is important that the run-time invariant subtypes have static bounds. The intent of the P1003.5 approach, which has been followed here, is to permit compile-time optimizations (including dead code elmination) based on the values of these bounds. This preserves an important functionality of the C-language interface.

As in P1003.5, all the limits are in one package. There are disadvantages to this choice. It creates a textual distance between the place where a limit is defined and the section of this standard where it applies. However, there were overriding reasons for putting them in one package, that does not depend on any of the other packages:

1. Several of these limits (such as {PAGESIZE}) apply to more than one optional part of this standard.

2. If an option is not supported, the implementation does not need to provide the corresponding Ada package specifications.

As in P1003.5, the limits are in the same package with other globally used declarations, in a package on which all of the other package specifications depend. There are several disadvantages:

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1. There is no logical need for dependence of the other package specifications on the values of these limits.

2. Constants declared in this package cannot be modified without recompilation of the other package specifications. This may be an impediment to providing accurate compile-time limits for installations of this interface on different systems.

A complete reorganization of the P1003.5 package `POSIX` seems called for, though it is beyond the scope of this document. P1003.1 and P1003.4 use separate header files (e.g. **errno.h, limits.h, time.h**) for the corresponding declarations. If this were done, all limits should probably be in a package `POSIX_Limits`, all error codes in a package `POSIX_Errors`, etc.

Another alternative considered was to replicate the declarations for the limits in each package to which they apply. The reason for replicating, rather than just putting the limit into one of the packages, is that if a limit is in the package corresponding to an option that is not supported, an application needing access to this limit for use with some other option (which is supported) would still depend on a package that is not supported. Replication did not seem attractive, because it would create visibility problems where multiple **use** clauses are in force, and because modifications to the limits would still require recompilation of the packages that contain them.

2.7.6 Blocking Behavior

On examination of the P1003.4a document, it appears that blocking operations of P1003.4 and P1003.1 are only *required* to block the calling thread, but an implementation may choose to block the entire process or any subset of the threads within the process. Therefore, the two blocking options of P1003.5 will not be sufficient for implementations over POSIX threads. There will be three possibilities:

1. The entire process is blocked, reliably.

2. Only the calling task is blocked, reliably.

3. At least the calling task is blocked, but not necessarily all the tasks in the process.

This issue should be addressed in the context of P1003.4a and P1003.5.

2.7.7 Operations on Addresses

The content of the `POSIX_Address_Arithmetic` package is based on recommendations UI-0065 of the ISO/IEC JTC1 SC22 WG9 Uniformity Rapporteur Group (URG). However the URG suggests that these should be in the `System` package. A separate package is provide here, for several reasons. First, this is felt to provide better upward compatibility of existing applications and Ada language implementations, which may provide for other forms of address arithmetic in `System`. Second, it appears that Ada 9X will provide some form of standard support for address arithmetic,

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

and one does not want to impose a requirement that will conflict with the Ada 9X solution.

2.7.8 Ada 9X Considerations

In Ada 9X, the packages described in this section could be declared as child library units. For example, `POSIX.Extension.Limits` could be a child of `POSIX.Extension`, which could in turn be a child of `POSIX`. It is not clear, however, whether this is desirable.

It is possible that Ada 9X will define a standard type which is similar to `Duration` but with finer precision and less range. A one-to-one mapping of values of this type to the nanosecond range used here might be an appropriate requirement. Likewise, Ada 9X may define a new real-time clock with its own time type. If so, it may be appropriate to require a one-to-mapping of this type and clock to corresponding entities in the POSIX interface.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

This side is intentionally blank.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Section 3: Revisions to Process Primitives

3.1 Signals

This section describes the extension to P1003.5 signals to support real-time requirements.

Specification of Package `POSIX_Signals`

```
with POSIX;                              -- from P1003.5
with POSIX_Process_Identification;   -- from P1003.5
with System;
package POSIX_Signals is
    -- Declarations carried forward from P1003.5 are not shown here.
      ...
    -- §3.1.1 Signal Concepts    ...
    -- §3.1.2 Signal Actions    ...
    -- §3.1.3 Wait for Queued Signals    ...
    -- §3.1.4 Queue a Signal to a Process    ...
private
    -- Declarations carried forward from P1003.5 are not shown here.
      ...
    -- impl.-defined
end POSIX_Signals;
```

3.1.1 Signal Concepts

3.1.1.1 Synopsis

```
    Signal_Bus_Error,
    SIGBUS: constant POSIX_Signals.Signal:= impl.-defined static const.;
    subtype Realtime_Signal is POSIX_Signals.Signal range impl.-defined;
    function Signal_Reference(Sig: POSIX_Signals.Signal) return System.Address;
    type Signal_Event is private;
    type Signal_Value is private;
    function Get_Signal(Event: Signal_Event) return POSIX_Signals.Signal;
    procedure Set_Signal
      (Event: in out Signal_Event;
       Sig:   in POSIX_Signals.Signal);
    function Get_Value(Event: Signal_Event) return Signal_Value;
    procedure Set_Value
      (Event: in out Signal_Event;
       Value: in Signal_Value);
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

3.1.1.2 Description

3.1.1.3 Signal Names

The Ada-language constant for the signal for access to an undefined portion of a memory object shall correspond to the constant of the P1003.4 C-Language Specification as shown in Table 3.1.

Table 3.1 – Memory Mapping Error Signal

C-Language	Ada-Language
SIGBUS	`SIGBUS`

The Ada-language names of the type and constants for real-time signals shall correspond to the data type and constants of the P1003.4 C-Language Specification as shown in Table 3.2.

Table 3.2 – Signal Numbers

C-Language	Ada-Language	Notes
{SIGRTMIN} {SIGRTMAX}	`Realtime_Signal'First` `Realtime_Signal'Last`	
int	`Signal`	(1)

NOTES:

(1) `POSIX_Signals.Signal` is defined in P1003.5.

The range `Realtime_Signal` shall include at least `POSIX.Portable_Realtime_Signal_Max` values, and shall not overlap with the signals defined in P1003.5.

`Signal_Reference` shall return a unique value of `System.Address` for each value of `POSIX_Signal` supported by the implementation. The value returned shall identify the corresponding signal, for use in an entry address clause to associate a handler with the signal, at task activation time. For each signal defined in P1003.5, the value returned by `Signal_Reference` shall be the same as the value of the corresponding address constant declared in `POSIX_Signals`. That is, if `Signal_`*XXX* is the name of a signal declared in `POSIX_Signals`, then `Signal_Reference(Signal_`*XXX*`) = POSIX_Signals.Signal_`*XXX*`_Ref`.

3.1.1.4 Signal Event Type

The Ada-language names for the signal event and signal event value types shall correspond to data types of the P1003.4 C-Language Specification as shown in Table 3.3.

Table 3.3 – Realtime Signal Event Type

C-Language	Ada-Language
sigevent	`Signal_Event`
sigval	`Signal_Value`

The `Signal_Value` type is private. Values of this type can be converted, using an

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

instantiation of `Unchecked_Conversion`, to other types. The size of this type must be large enough to hold a value of type `Integer` or type `System.Address`.

The Ada-language names for read and write operations of components of the `Signal_Event` type shall correspond to components of the *sigevent* type of the P1003.4 C-Language Specification as shown in Table 3.4.

Table 3.4 – Signal Event Operations

C-Language Member	Ada-Language Get Value	Ada-Language Set Value
sigev_signo	`Get_Signal`	`Set_Signal`
sigev_value	`Get_Value`	`Set_Value`

`Get_Signal` returns the signal associated with `Event`.

`Set_Signal` sets the signal associated with `Event` to the the value `Sig`.

`Get_Value` returns the value associated with `Event`.

`Set_Value` sets the value associated with `Event` to the the specified value.

3.1.1.5 Error Handling

`POSIX_Error` shall be raised and the error code shall be as specified below, for the specified conditions.

`Invalid_Argument`
: `Signal_Reference` is called with a value of `Sig` that is not a valid signal, or is a signal for which the implementation does not support user-defined signal handlers.

No other error conditions are specified for these operations.

3.1.1.6 References

SIGBUS, P1003.4 § 3.3.1.1; {SIGRTMIN}, P1003.4 § 3.3.1.1; {SIGRTMAX}, P1003.4 § 3.3.1.1; *sigevent*, P1003.4 § 3.3.1.2.

3.1.2 Signal Actions

3.1.2.1 Synopsis

```
type Signal_Source is range impl.-defined static range;
   From_User:          constant Signal_Source:= impl.-defined;
   From_Queue_Signal:  constant Signal_Source:= impl.-defined;
   From_Timer:         constant Signal_Source:= impl.-defined;
   From_Async_IO:      constant Signal_Source:= impl.-defined;
   From_Message_Queue: constant Signal_Source:= impl.-defined;
type Signal_Info is private;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
   function Get_Signal(Info: Signal_Info) return POSIX_Signals.Signal;
   procedure Set_Signal
     (Info: in out Signal_Info;
      Sig:  in POSIX_Signals.Signal);
   function Get_Source(Info: Signal_Info) return Signal_Source;
   procedure Set_Source
     (Info:   in out Signal_Info;
      Source: in Signal_Source);
   function Has_Value(Source: Signal_Source) return Boolean;
   function Get_Value(Info: Signal_Info) return Signal_Value;
   procedure Set_Value
     (Info:  in out Signal_Info;
      Value: in Signal_Value);
```

3.1.2.2 Description

The Ada-language names for the signal information and associated types shall correspond to data types of the P1003.4 C-Language Specification as shown in Table 3.5.

Table 3.5 – Realtime Signal Information Type

C-Language	Ada-Language
siginfo_t	Signal_Info
int	Signal
int	Signal_Source

The Ada-language names for the constants of the Signal_Source type shall correspond to constants of the P1003.4 C-Language Specification as shown in Table 3.6.

Table 3.6 – Signal Sources

C-Language	Ada-Language
SI_USER	From_User
SI_QUEUE	From_Queue_Signal
SI_TIMER	From_Timer
SI_ASYNCIO	From_Async_IO
SI_MESGQ	From_Message_Queue

The Ada-language names for read and write operations of components of the Signal_Info type shall correspond to components of the *siginfo_t* type of the P1003.4 C-Language Specification as shown in Table 3.7.

Table 3.7 – Signal Information Operations

C-Language Member	Ada-Language	
	Get Value	Set Value
si_signo	Get_Signal	Set_Signal
si_code	Get_Source	Set_Source
si_value	Get_Value	Set_Value

Values of Signal_Source are used to identify the source of a signal.

Signal_Info is used to pass information about the event associated with a signal occurrence.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

149 `Get_Signal` returns the signal associated with `Info`.

150 `Set_Signal` sets the signal associated with `Info` to the the value `Sig`.

151 `Get_Source` returns the signal source associated with `Info`.

152 `Set_Source` sets the signal source associated with `Info` to the value `Source`.

153 `Get_Value` returns the value associated with `Info`.

154 `Set_Value` sets the value associated with `Info` to the specified value.

155 `Has_Value` shall return `True` if and only if `Source` is one of the signal sources for
156 which the *siginfo_t* parameter of a signal handler (see 3.1.2.5) has the contents of
157 *si_value* defined. The P1003.4 C-Language Specification specifies that this includes
158 at least SI_QUEUE, SI_TIMER, SI_ASYNCIO, and SI_MESGQ.

159 3.1.2.3 Error Handling

160 No error conditions are specified for these operations.

161 3.1.2.4 References

162 SI_USER, SI_QUEUE, SI_TIMER, SI_ASYNCIO, SI_MESGQ, P1003.4 § 3.3.1.3;
163 *siginfo_t*, P1003.4 § 3.3.1.3; signal handler, P1003.4 § 3.3.1.3.

164 3.1.2.5 Realtime Signal Handler Entries

165 This standard specifies extensions to the facilities for handling signals specified by
166 P1003.5.

167 A handler for a realtime signal may be specified by binding the signal to an entry of a
168 task, using an Ada address clause. The parameter profile of a handler for a realtime
169 signal may be of the form shown below.

```
170 entry Entry_Name
171    (Sig:     in POSIX_Signals.Signal;
172     Info:    in Signal_Info;
173     Context: in Access_Type);
```

174 The delivery of signals to such an entry shall be as specified in the P1003.4 C-
175 Language Specification for a handler bound by *sigaction* with flag SA_SIGINFO.

176 The compiler shall not accept an entry address clause for an entry with the form spec-
177 ified above unless the package `POSIX_Signals` is in the context of the compilation
178 unit.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

3.1.3 Wait for Queued Signals

3.1.3.1 Synopsis

```
procedure Await_Signal
   (Set:            in POSIX_Signals.Signal_Set;
    Info:           in out Signal_Info;
    Masked_Signals: in POSIX.Signal_Masking := POSIX.RTS_Signals);
procedure Await_Signal_Or_Timeout
   (Set:            in POSIX_Signals.Signal_Set;
    Info:           in out Signal_Info;
    Timeout:        in POSIX.Timespec;
    Masked_Signals: in POSIX.Signal_Masking := POSIX.RTS_Signals);
```

3.1.3.2 Description

The Ada-language operations for awaiting a signal, with and without timeout, shall correspond to the P1003.4 C-Language Specification operations as shown in Table 3.8.

Table 3.8 – Wait for Queued Signals

C-Language			Ada-Language			Notes
sigwaitrt			Await_Signal			
	set	*(I)*		Set	*(I)*	
	info	*(I)*		Value	*(I)*	
	(none)	*(I)*		Masked_Signals	*(I)*	(1)
sigtimedwait			Await_Signal_Or_Timeout			
	set	*(I)*		Set	*(I)*	
	info	*(I)*		Value	*(I)*	
	timeout	*(I)*		Timeout	*(I)*	
	(none)	*(I)*		Masked_Signals	*(I)*	(1)

NOTES:

(1) The Ada parameter Masked_Signals shall specify the set of signals that are to be added to the signal mask that is in effect during the operation. The previous signal mask shall be restored when the operation completes.

3.1.3.3 References

sigwaitrt and *sigtimedwait*, P1003.4 § 3.3.8.

3.1.4 Queue a Signal to a Process

3.1.4.1 Synopsis

```
procedure Queue_Signal
   (Process: in POSIX_Process_Identification.Process_ID;
    Sig:     in POSIX_Signals.Signal);
procedure Queue_Signal
   (Process: in POSIX_Process_Identification.Process_ID;
    Sig:     in POSIX_Signals.Signal;
    Value:   in Signal_Value);
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

3.1.4.2 Description

The Ada-language operations for queuing a signal to a process shall correspond to the P1003.4 C-Language Specification operations as shown in Table 3.9.

Table 3.9 – Queue a Signal to a Process

C-Language			Ada-Language			Notes
sigqueue			`Queue_Signal`			
	pid	*(I)*		`Process`	*(I)*	(1)
	signo	*(I)*		`Signo`	*(I)*	
	value (= 0)	*(I)*		*(none)*	*(I)*	
sigqueue			`Queue_Signal`			
	pid	*(I)*		`Process`	*(I)*	(1)
	signo	*(I)*		`Sig`	*(I)*	
	value (≠ 0)	*(I)*		`Value`	*(I)*	

NOTES:

(1) The case *pid* = 0 corresponds to the case `Process` = `Null_Process_ID`.

3.1.4.3 Error Handling

No error conditions are specified for these operations.

3.1.4.4 References

sigqueue, P1003.4 § 3.3.9.

3.2 Rationale Relating to Realtime Signals Extension

3.2.1 Sigwaitrt and Sigtimedwait

The *sigwaitrt* and *sigtimedwait* operations cannot be mapped to Ada tasking operations, such as accept statements, since they block the whole process unless P1003.4a is supported.

3.2.2 Signal Handler Profiles

There seems to be a gap in the P1003.5 document, that it does not specify any requirement on the parameter profile of an entry that is bound to a signal. The present binding provides such a profile only for realtime signals.

In the C-language interface, the *sigval* component of a *sigevent* structure is large enough to hold a pointer or integer value. In this Ada interface, it is required that it be large enough to hold an integer or address. It is not required to be large enough to hold an Ada access value, because such values may be longer than C pointers, due to extra space that may be required to represent a constraint. Perhaps this requirement should be tightened, so that at least constrained access types are allowed.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

3.2.3 Composability Considerations

In general, integrating subsystems requires some discipline and coordination in development. Pthreads introduces new points where coordination is required. One of these is signals. Signals are a non-sharable resource. In general it is not possible to compose subsystems that use signals, unless it can be determined that they do not use the same signals. This is true regardless of what language those subsystems are written in. It is especially likely to be a problem where off-the-shelf subsystems (libraries) are linked with an Ada program. In the case that the off-the-shelf subsystem is written in C, the problem is more likely, since there is no check that the C code does not use the signals reserved for the Ada language implementation (e.g. SIGALRM, for time-outs). There is also a potential problem if a C-language subsystem calls an Ada-language subprogram, which in turn is interrupted by a C-language signal handler or calls a C-language subprogram. In this case, if the C-language subprogram calls *longjmp*, storage may be lost and Ada runtime system data structures may be left in a chaotic state.

3.2.4 P1003.4a Considerations

P1003.4a complicates the signal model. It introduces per-thread signal mask, and the possibility of directing a signal to a specific thread (rather than just to a whole process). The P1003.5 model only allows a subset of these possibilities, i.e. interrupt directed to the entire process and delivery to tasks.

P1003.4a also removes the possibility of blocking and unblocking a signal for all the tasks of a process, via an operation with global effect. This is unlike the P1003.5 model.

The P1003.5 model is the basis for this Ada-language binding to P1003.4. If the model is extended to support all the options of P1003.4a, the real-time signal interface described here will need to be revisited.

3.2.5 P1003.5 Considerations

The present P1003.5 interface, after which this one is modeled, prevents the application from using the full functionality of POSIX signals. In particular, there is no way for an application to trap a signal that is handled by the Ada RTS. In principle, a mechanism could be provided, whereby either: (1) the Ada RTS catches the signal, and then optionally passes it on to the application; (2) the application catches the signal, and the RTS signal handler is invoked on return from the application handler.

3.2.6 Ada 9X Issues

3.2.6.1 Signal Handler Model

In [8], the preferred model for handling interrupts is via protected procedures:

```
package Interrupt_Management is
   type Interrupt_ID is impl.-defined;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
   Reserved_Interrupt: exception;
   function Is_Reserved (Interrupt : Interrupt_ID) return Boolean;
   type Handler_Type is access protected procedure;
   procedure Attach_Handler
      (Handler:   in Handler_Type;
       Interrupt: in Interrupt_ID);
   procedure Detach_Handler
      (Interrupt: in Interrupt_ID);
   function Is_Attached(Interrupt: Interrupt_ID) return Boolean;
   function Current_Handler(Interrupt: Interrupt_ID) return Handler_Type;
end Interrupt_Management;
```

This package cannot be applied directly to POSIX signals, since `Handler_Type` type does not have any parameters. For Ada 9X, it seems POSIX should provide a similar package:

```
with POSIX_Signals;                          -- from P1003.5
package POSIX_Signal_Management is
   Reserved_Signal: exception;
   function Is_Reserved(Sig: POSIX_Signals.Signal) return Boolean;
   type Context_Type is impl.-defined;
   type Handler_Type is
      access protected procedure
         (Sig:     in POSIX_Signals.Signal;
          Info:    in Signal_Info;
          Context: in Context_Type);
   procedure Attach_Handler
      (Handler: in Handler_Type;
       Sig:     in POSIX_Signals.Signal);
   procedure Detach_Handler
      (Sig: in POSIX_Signals.Signal);
   function Is_Attached(Sig: POSIX_Signals.Signal) return Boolean;
   function Current_Handler(Sig: POSIX_Signals.Signal) return Handler_Type;
end POSIX_Signal_Management;
```

The semantics of this package would be analogous to the standard package `Interrupt_Management`.

3.2.6.2 Signal Priorities

In [8], it is proposed that masking of interrupts be required during execution of an interrupt handler and during execution of operations on protected records with sufficiently high ceiling priorities. This is essential to enforce mutual exclusion between interrupt handlers and other users of protected records. Raising a task's priority is also the general Ada 9X way of masking out interrupts. To implement the Ada 9X model of protected record locking and interrupt handling, it would be necessary to either mask all signals whenever a protected procedure with interrupt priority is executing, or to impose an implementation-defined priority structure on signals and only mask signals up to a certain priority level.

Note that P1003.4 recognizes interrupt priorities only in a different sense: for arbitrating the order of delivery of signals that are simultaneously pending.

A possible solution to the problem of masking specific signals during protected op-

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

erations is to define a pragma `Mask_Signals`, which would cause a specified set of signals to be masked during every protected operation of a protected record.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Section 4: Revisions to Process Environment

4.8 Configurable System Variables

Specification of Package `POSIX_Configurable_System_Limits`

The package `POSIX_Configurable_System_Limits` contains functions that may be used to obtain the values of limits imposed on the use of the interfaces specified by this standard that are imposed by the particular implementation.

```
with POSIX;                              -- from P1003.5
package POSIX_Configurable_System_Limits is
   -- Declarations carried forward from P1003.5 are not shown here.
     ...
   -- §4.8.1 Get Configurable System Variables   ...
end POSIX_Configurable_System_Limits;
```

The function to query whether an option is supported, whose name is of the form `POSIX_`*XXX*`_[Is/Are]_Supported`, returns a value of the subtype `POSIX.POSIX_`–*XXX*`_Support`. If the function returns `True`, the option shall be supported. If the function returns `False`, whether the option is supported is unspecified.

4.8.1 Get Configurable System Variables

4.8.1.1 Synopsis

```
  function Async_IO_Is_Supported
     return POSIX.Async_IO_Support;
  function File_Sync_Is_Supported
     return POSIX.File_Sync_Support;
  function Memory_Locking_Is_Supported
     return POSIX.Memory_Locking_Support;
  function Memory_Range_Locking_Is_Supported
     return POSIX.Memory_Range_Locking_Support;
  function Memory_Protection_Is_Supported
     return POSIX.Memory_Protection_Support;
  function Message_Passing_Is_Supported
     return POSIX.Message_Passing_Support;
  function Mapped_Files_Are_Supported
     return POSIX.Mapped_Files_Support;
  function Prioritized_IO_Is_Supported
     return POSIX.Prioritized_IO_Support;
  function Priority_Scheduling_Is_Supported
     return POSIX.Priority_Scheduling_Support;
  function Realtime_Files_Are_Supported
     return POSIX.Realtime_Files_Support;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
   function Realtime_Signals_Are_Supported
      return POSIX.Realtime_Signals_Support;
   function Semaphores_Are_Supported
      return POSIX.Semaphores_Support;
   function Shared_Memory_Objects_Are_Supported
      return POSIX.Shared_Memory_Objects_Support;
   function Sync_IO_Is_Supported
      return POSIX.Sync_IO_Support;
   function Timers_Are_Supported
      return POSIX.Timers_Support;
   function Page_Size
      return POSIX.Page_Size_Range;
   function List_IO_Max
      return POSIX.List_IO_Maxima;
   function Async_IO_Max
      return POSIX.Async_IO_Maxima;
   function Async_IO_Priority_Delta_Max
      return Natural;
   function Timer_Overruns_Max
      return POSIX.Timer_Overruns_Maxima;
   function Message_Queue_Open_Max
      return POSIX.Message_Queue_Open_Maxima;
   function Message_Queue_Priority_Max
      return POSIX.Message_Queue_Priority_Maxima;
   function Realtime_Signals_Max
      return POSIX.Realtime_Signals_Maxima;
   function Semaphores_Max
      return POSIX.Semaphores_Maxima;
   function Semaphores_Value_Max
      return POSIX.Semaphores_Value_Maxima;
   function Queued_Signals_Max
      return POSIX.Queued_Signals_Maxima;
   function Timers_Max
      return POSIX.Timers_Maxima;
```

4.8.1.2 Description

The Ada-language functions for obtaining the values of configurable system variables shall correspond to configurable system variables of the P1003.4 C-Language Specification, as shown in Table 4.1.

As specified in § 2.4.2, the values reported by these functions shall reflect the actual limits as they apply to the Ada application.

Note that the P1003.4 C-Language Specification does not define a configurable system variable for {AIO_PRIO_DELTA_MAX} or {INTERVAL_MAX}.

4.8.1.3 Error Handling

None of these functions shall raise any exception.

4.8.1.4 References

options, P1003.4 § 2.9.3.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 4.1 – Functions for Configurable System Variables

C-Language	Ada-Language
{_SC_ASYNCHRONOUS_IO}	Async_IO_Is_Supported
{_SC_FSYNC}	File_Sync_Is_Supported
{_SC_MAPPED_FILES}	Mapped_Files_Are_Supported
{_SC_MEMLOCK}	Memory_Locking_Is_Supported
{_SC_MEMLOCK_RANGE}	Memory_Range_Locking_Is_Supported
{_SC_MEMORY_PROTECTION}	Memory_Protection_Is_Supported
{_SC_MESSAGE_PASSING}	Message_Passing_Is_Supported
{_SC_PRIORITIZED_IO}	Prioritized_IO_Is_Supported
{_SC_PRIORITY_SCHEDULING}	Priority_Scheduling_Is_Supported
{_SC_REALTIME_FILES}	Realtime_Files_Are_Supported
{_SC_REALTIME_SIGNALS}	Realtime_Signals_Are_Supported
{_SC_SEMAPHORES}	Semaphore_Is_Supported
{_SC_SHARED_MEMORY_OBJECTS}	Shared_Memory_Objects_Are_Supported
{_SC_SYNCHRONIZED_IO}	Sync_IO_Is_Supported
{_SC_TIMERS}	Timers_Are_Supported
{_SC_PAGESIZE}	Page_Size
{_SC_AIO_LISTIO_MAX}	List_IO_Max
{_SC_AIO_MAX}	Async_IO_Max
{_SC_AIO_PRIO_DELTA_MAX}	Async_IO_Priority_Delta_Max
{_SC_DELAYTIMER_MAX}	Timer_Overruns_Max
{_SC_MQ_OPENMAX}	Message_Queue_Open_Max
{_SC_MQ_PRIO_MAX}	Message_Queue_Priority_Max
{_SC_RTSIG_MAX}	Realtime_Signals_Max
{_SC_SEM_NSEMS_MAX}	Semaphores_Max
{_SC_SEM_VALUE_MAX}	Semaphores_Value_Max
{_SC_SIGQUEUE_MAX}	Queued_Signals_Max
{_SC_TIMER_MAX}	Timers_Max

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.9 Rationale

4.9.1 Packaging

To follow the structure of the P1003.1 and P1003.4 documents, the declarations of the functions for configurable system have been inserted in the P1003.5 package `POSIX_Configurable_System_Limits`.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Section 5: Revisions to Files and Directories

5.6.1 File Characteristics: Header and Data Structure

Specification of Package POSIX_File_Status

```
package POSIX_File_Status is
    -- Declarations carried forward from P1003.5 are not shown here.
      ...
    -- §5.6.2 File Characteristics: Header and Data Structure   ...
end POSIX_File_Status;
```

Specification of Package POSIX_Files

```
with POSIX;                          -- from P1003.5
with POSIX_IO;                       -- from P1003.5
with POSIX_Permissions;              -- from P1003.5
package POSIX_Files is
    -- Declarations carried forward from P1003.5 are not shown here.
      ...
    -- §5.6.4 Change File Modes   ...
    -- §5.6.7 Truncate File to A Specified Length   ...
end POSIX_Files;
```

5.6.2 File Characteristics: Header and Data Structure

5.6.2.1 Synopsis

```
   function Is_Realtime_File(File_Status: POSIX_File_Status.Status)
      return Boolean;
   function Is_Shared_Memory(File_Status: POSIX_File_Status.Status)
      return Boolean;
   function Is_Message_Queue(File_Status: POSIX_File_Status.Status)
      return Boolean;
   function Is_Semaphore(File_Status: POSIX_File_Status.Status)
      return Boolean;
```

5.6.2.2 Description

The Ada-language operations to determine whether a file status value indicates a semaphore, message queue, or realtime file shall correspond to the P1003.4 C-Language Specification test macros as shown in Table 5.1.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 5.1 – File Characteristics: Header and Data Structure

C-Language	Ada-Language
S_TYPEISSEM	Is_Semaphore
S_TYPEISSHM	Is_Shared_Memory
S_TYPEISMQ	Is_Message_Queue
S_TYPEISRTFILE	Is_Realtime_File

5.6.2.3 References

S_TYPEISSEM, S_TYPEISMQ, S_TYPEISRTFILE, P1003.4 §5.6.1.

5.6.4 Change File Modes

5.6.4.1 Synopsis

```
procedure Change_Permissions
   (File:       in  POSIX_IO.File_Descriptor;
    Permission: in  POSIX_Permissions.Permission_Set);
```

5.6.4.2 Description

The Ada operation for changing the permissions of a file shall correspond to the P1003.4 C-Language Specification operations as shown in Table 5.2.

Table 5.2 – Change File Modes

C-Language		Ada-Language	
fchmod		Change_Permissions	
fildes	*(I)*	File	*(I)*
mode	*(I)*	Permission	*(I)*

5.6.4.3 References

fchmod, P1003.4 §5.6.4.

5.6.7 Truncate File to A Specified Length

5.6.7.1 Synopsis

```
procedure Truncate_File
   (File:   in  POSIX_IO.File_Descriptor;
    Length: in  POSIX.IO_Count);
```

5.6.7.2 Description

The Ada operation for truncating a file to a specified length shall correspond to the P1003.4 C-Language Specification operation as shown in Table 5.3.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 5.3 – Truncate File to A Specified Length

C-Language		Ada-Language	
ftruncate		`Truncate_File`	
fildes	*(I)*	`File`	*(I)*
length	*(I)*	`Length`	*(I)*

5.6.7.3 References

ftruncate, P1003.4 § 5.6.7.

5.7 Configurable Pathname Variables

Specification of Package `POSIX_Configurable_File_Limits`

```
with POSIX;                          -- from P1003.5
with POSIX_IO;                       -- from P1003.5
with POSIX_Permissions;              -- from P1003.5
package POSIX_Configurable_File_Limits is
   -- Declarations carried forward from P1003.5 are not shown here.
     ...
   -- § 5.7.1 Get Configurable Pathname Variables   ...
end POSIX_Configurable_File_Limits;
```

5.7.1 Get Configurable Pathname Variables

5.7.1.1 Synopsis

```
   function Sync_IO_Is_Supported
      (Name: POSIX.Pathname)
     return Boolean;
   function Sync_IO_Is_Supported
      (File: POSIX_IO.File_Descriptor)
     return Boolean;
   function Async_IO_Is_Supported
      (Name: POSIX.Pathname)
     return Boolean;
   function Async_IO_Is_Supported
      (File: POSIX_IO.File_Descriptor)
     return Boolean;
   function Prioritized_IO_Is_Supported
      (Name: POSIX.Pathname)
     return Boolean;
   function Prioritized_IO_Is_Supported
      (File: POSIX_IO.File_Descriptor)
     return Boolean;
```

5.7.1.2 Description

The Ada-language functions for obtaining the values of configurable system variables shall correspond to configurable system variables of the P1003.4 C-Language

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Specification, as shown in Table 5.4.

Table 5.4 – Get Configurable Pathname Variables

C-Language		Ada-Language	
pathconfig		`Async_IO_Is_Supported`	
path	*(I)*	`Name`	*(I)*
name (= _PC_ASYNC_IO)	*(I)*	*(none)*	*(I)*
fpathconfig		`Async_IO_Is_Supported`	
fildes	*(I)*	`File`	*(I)*
name (= _PC_ASYNC_IO)	*(I)*	*(none)*	*(I)*
pathconfig		`Sync_IO_Is_Supported`	
path	*(I)*	`Name`	*(I)*
name (= _PC_SYNC_IO)	*(I)*	*(none)*	*(I)*
fpathconfig		`Sync_IO_Is_Supported`	
fildes	*(I)*	`File`	*(I)*
name (= _PC_SYNC_IO)	*(I)*	*(none)*	*(I)*
pathconfig		`Prioritized_IO_Is_Supported`	
path	*(I)*	`Name`	*(I)*
name (= _PC_PRIO_IO)	*(I)*	*(none)*	*(I)*
fpathconfig		`Prioritized_IO_Is_Supported`	
fildes	*(I)*	`File`	*(I)*
name (= _PC_PRIO_IO)	*(I)*	*(none)*	*(I)*

5.7.1.3 References

_PC_SYNC_IO, _PC_ASYNC_IO, _PC_PRIO_IO, P1003.4 § 5.7.2; *pathconfig* and *fpathconfig*, P1003.1 § 5.7.1, P1003.4 § F 2.2.

5.8 Rationale for Revisions to Files and Directories

5.8.1 Packaging

The declarations specified in this section are additions to packages specified by P1003.5.

Note that the P1003.5 package `POSIX_Configurable_File_Limits` needs updating anyway, to support the P1003.1 *fpathconfig* capability, which makes it dependent on `POSIX_IO`.

P1003.1a as well as P1003.4 defines the *fsync* operation, but the P1003.1a operations are not yet covered by P1003.5. We have tentatively placed *fsync* in `POSIX_Files`.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Section 6: Revisions to Input and Output Primitives

Specification of Package POSIX_IO

```
with POSIX;                              -- from P1003.5
with POSIX_Permissions;                  -- from P1003.5
package POSIX_IO is
   -- §6.5.1 Data Definitions for File Control Operations   ...
   -- §6.6.1 Synchronize a File's State   ...
   -- §6.6.2 Data Synchronization   ...
end POSIX_IO;
```

6.5 Control Operations on Files

6.5.1 Data Definitions for File Control Operations

6.5.1.1 Synopsis

```
   File_Synchronized: constant POSIX_IO.Open_Option_Set:= impl.-defined;
   Data_Synchronized: constant POSIX_IO.Open_Option_Set:= impl.-defined;
```

6.5.1.2 Description

P1003.4 defines additional values of the type POSIX_IO.Open_Option_Set. These additional options may be used with the Open, Open_Or_Create, Get_File_Control, and Set_File_Control operations defined by P1003.5. The Ada-language names of the constants shall correspond to the constants of the P1003.4 C-Language Specification as shown in Table 6.1.

Table 6.1 – Open Option Sets

C-Language	Ada-Language
O_DSYNC	Data_Synchronized
O_SYNC	File_Synchronized

6.5.1.3 References

O_SYNC and O_DSYNC, P1003.4 §5.3.1, P1003.4 §6.5.1.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.6 File Synchronization

This section describes the facilities for synchronized input and output available under this standard.

6.6.1 Synchronize a File's State

6.6.1.1 Synopsis

```
procedure Synchronize_File(File:  in POSIX_IO.File_Descriptor);
```

6.6.1.2 Description

The Ada-language operation to force logical completion of all I/O operations on a file shall correspond to operations of the P1003.4 C-Language Specification as shown in Table 6.2.

Table 6.2 – Synchronize a File's State

C-Language	Ada-Language
fsync *fildes* (I)	Synchronize_File File (I)

6.6.1.3 References

fsync, P1003.4 § 6.6.1.

6.6.2 Data Synchronization

6.6.2.1 Synopsis

```
procedure Synchronize_Data(File:  in POSIX_IO.File_Descriptor);
```

6.6.2.2 Description

The Ada-language operation to force data completion of all I/O operations on a file, including flushing of any output buffers to the peripheral device, shall correspond to operations of the P1003.4 C-Language Specification as shown in Table 6.3.

Table 6.3 – Data Synchronization

C-Language	Ada-Language
fdatasync *fildes* (I)	Synchronize_Data File (I)

6.6.2.3 References

fdatasync, P1003.4 § 6.6.2.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.7 Asynchronous Input and Output

This section describes the facilities for asynchronous input and output available under this standard.

Specification of Package POSIX_Asynchronous_IO

```
with POSIX;                                -- from P1003.5
with POSIX_IO;                             -- from P1003.5
with POSIX_Signals;                        -- from P1003.5
package POSIX_Asynchronous_IO is
   -- §6.7.0 Data Definitions for Asynchronous Input and Output ...
   -- §6.7.1 Asynchronous Read ...
   -- §6.7.2 Asynchronous Write ...
   -- §6.7.3 List Directed I/O ...
   -- §6.7.4 Retrieve Error Status of Asynchronous I/O Operation ...
   -- §6.7.5 Retrieve Return Status of Asynchronous I/O Operation ...
   -- §6.7.6 Cancel Asynchronous I/O Request ...
   -- §6.7.7 Wait for Asynchronous I/O Request ...
   -- §6.7.8 Asynchronous File Synchronization ...
private
   -- impl.-defined;
end POSIX_Asynchronous_IO;
```

6.7.0 Data Definitions for Asynchronous Input and Output

6.7.0.1 Synopsis

```
   type Control_Block is private;
   type Control_Block_Pointer is access Control_Block;
   type Control_Block_List is
      array (Positive range <>) of Control_Block_Pointer;
   type List_IO_Operations is
      (Read,
       Write,
       No_Op);
   type IO_Buffer_Pointer is access POSIX_IO.IO_Buffer;
   function Get_File(CB: Control_Block_Pointer)
     return POSIX_IO.File_Descriptor;
   procedure Set_File
      (CB:    in out Control_Block_Pointer;
       File: in POSIX_IO.File_Descriptor);
   function Get_Offset(CB: Control_Block_Pointer) return POSIX_IO.IO_Offset;
   procedure Set_Offset
      (CB:    in out Control_Block_Pointer;
       Value: in POSIX_IO.IO_Offset);
   function Get_Buffer(CB: Control_Block_Pointer) return IO_Buffer_Pointer;
   procedure Set_Buffer
      (CB:    in out Control_Block_Pointer;
       Value: in IO_Buffer_Pointer);
   function Get_Length(CB: Control_Block_Pointer) return POSIX.IO_Count;
   procedure Set_Length
      (CB:    in out Control_Block_Pointer;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        Value: in POSIX.IO_Count);
   function Get_Priority(CB: Control_Block_Pointer) return Natural;
   procedure Set_Priority
      (CB:    in out Control_Block_Pointer;
       Value: in Natural);
   function Get_Signal_Event(CB: Control_Block_Pointer)
     return POSIX_Signals.Signal_Event;
   procedure Set_Signal_Event
      (CB:    in out Control_Block_Pointer;
       Value: in POSIX_Signals.Signal_Event);
   function Get_IO_Operation(CB: Control_Block_Pointer)
     return List_IO_Operations;
   procedure Set_IO_Operation
      (CB:    in out Control_Block_Pointer;
       Value: in List_IO_Operations);
```

6.7.0.2 Description

The Ada-language names for data types associated with asynchronous IO shall correspond to data types of the P1003.4 C-Language Specification as shown in Table 6.4.

Table 6.4 – Asynchronous I/O Types

C-Language	Ada-Language	Notes
aiocb	`Control_Block`	
aiocb *	`Control_Block_Pointer`	
off_t	`IO_Offset`	(1)
size_t	`IO_Offset`	(1)
sigevent	`Signal_Event`	(2)
int	`List_IO_Operations`	
int	`Cancelation_Status`	
volatile void *	`IO_Buffer_Pointer`	

NOTES:

(1) `IO_Offset` is defined in the `POSIX` package.

(2) `Signal_Event` is defined in the `POSIX_Signals` package.

The Ada-language names for constants of the `List_IO_Operation` type shall correspond to constants of the P1003.4 C-Language Specification as shown in Table 6.5.

Table 6.5 – List I/O Operations

C-Language	Ada-Language
LIO_READ	`Read`
LIO_WRITE	`Write`
LIO_NOP	`No_Op`

The Ada-language names for read and write operations of the `Control_Block_Pointer` type shall correspond to components of the P1003.4 C-Language Specification *aiocb* type as shown in Table 6.6.

Objects of type `Control_Block_Pointer` are used to refer to asynchronous I/O control blocks. Control blocks are used to request asynchronous input and output operations and to determine the status of pending asynchronous input and output

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 6.6 – IO Control Block Operations

C-Language Member	Ada-Language Get Value	Ada-Language Set Value
aio_fildes	Get_File	Set_File
aio_offset	Get_Offset	Set_Offset
aio_buf	Get_Buffer	Set_Buffer
aio_nbytes	Get_Length	Set_Length
aio_reqprio	Get_Priority	Set_Priority
aio_sigevent	Get_Signal_Event	Set_Signal_Event
aio_lio_opcode	Get_IO_Operation	Set_IO_Operation

operations.

The application is responsible for the allocation and deallocation of control blocks. The effect of deallocating an I/O buffer or an asynchronous I/O control block before the I/O operation has completed and the return status has been retrieved is undefined.

6.7.0.3 References

aiocb, P1003.4 § 6.7.1.

6.7.1 Asynchronous Read

6.7.1.1 Synopsis

```
procedure Read(CB: in Control_Block_Pointer);
```

6.7.1.2 Description

The Ada-language operation for an asynchronous read operation shall correspond to the P1003.4 C-Language Specification operation as shown in Table 6.7.

Table 6.7 – Asynchronous Read

C-Language	Ada-Language
aio_read	Read
aiocbp (I)	CB *(I)*

6.7.1.3 Error Handling

In addition to the error conditions specified in the P1003.4 C-Language Specification, `Constraint_Error` shall be raised if the value of the `CB` parameter is **`null`**.

6.7.1.4 References

aio_read, P1003.4 § 6.7.2.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.7.2 Asynchronous Write

6.7.2.1 Synopsis

```
procedure Write(CB: in Control_Block_Pointer);
```

6.7.2.2 Description

The Ada-language operation for an asynchronous write operation shall correspond to the P1003.4 C-Language Specification operation as shown in Table 6.8.

Table 6.8 – Asynchronous Write

C-Language	Ada-Language
aio_write	Write
aiocbp (I)	CB (I)

6.7.2.3 Error Handling

In addition to the error conditions specified in the P1003.4 C-Language Specification, `Constraint_Error` shall be raised if the value of the `CB` parameter is **`null`**.

6.7.2.4 References

aio_write, P1003.4 § 6.7.3.

6.7.3 List Directed I/O

6.7.3.1 Synopsis

```
procedure List_IO_No_Wait
   (List:  in out Control_Block_List;
    Event: in POSIX_Signals.Signal_Event);
procedure List_IO_Wait
   (List:           in out Control_Block_List;
    Event:          in POSIX_Signals.Signal_Event;
    Masked_Signals: in POSIX.Signal_Masking := POSIX.RTS_Signals);
```

6.7.3.2 Description

The Ada-language operations for blocking and nonblocking list-directed I/O shall correspond to the P1003.4 C-Language Specification operation as shown in Table 6.9.

6.7.3.3 Error Handling

In addition to the error conditions specified in the P1003.4 C-Language Specification, `Constraint_Error` shall be raised if the value of the `CB` parameter is **`null`**.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 6.9 – List Directed I/O

C-Language			Ada-Language			Notes
lio_listio			List_IO_No_Wait			
	mode (= LIO_NOWAIT)	*(I)*		*(none)*	*(I)*	
	list	*(I)*		List	*(I)*	
	nent	*(I)*		List'length	*(I)*	
	sig	*(I)*		Event	*(I)*	
lio_listio			List_IO_Wait			
	mode (= LIO_WAIT)	*(I)*		*(none)*	*(I)*	
	list	*(I)*		List	*(I)*	
	nent	*(I)*		List'length	*(I)*	
	sig	*(I)*		Event	*(I)*	
	(none)	*(I)*		Masked_Signals	*(I)*	(1)

NOTES:

(1) `Masked_Signals` shall specify the set of signals that are to be added to the signal mask that is in effect during the operation. The previous signal mask shall be restored when the operation completes.

6.7.3.4 References

lio_listio, P1003.4 § 6.7.4.

6.7.4 Retrieve Error Status of Asynchronous I/O Operation

6.7.4.1 Synopsis

```
function Get_Error_Code
   (CB: in Control_Block_Pointer)
  return POSIX.Error_Code;
```

6.7.4.2 Description

The Ada-language operation for obtaining the error code associated with an asynchronous I/O operation shall correspond to the P1003.4 C-Language Specification operation as shown in Table 6.10.

Table 6.10 – Retrieve Error Status of Asynchronous I/O Operation

C-Language			Ada-Language		
aio_error			Get_Error_Code		
	aiocbp	*(I)*		CB	*(I)*
	<return-value>			*<return-value>*	

The `Get_Error_Code` return value `No_Error` corresponds to the *aio_error* return value zero.

6.7.4.3 Error Handling

In addition to the error conditions specified in the P1003.4 C-Language Specification, `Constraint_Error` shall be raised if the value of the `CB` parameter is **`null`**.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.7.4.4 References

aio_error, P1003.4 §6.7.5.

6.7.5 Retrieve Return Status of Asynchronous I/O Operation

6.7.5.1 Synopsis

```
function Characters_Transferred
   (CB: in Control_Block_Pointer) return POSIX.IO_Count;
```

6.7.5.2 Description

The Ada-language operation for determining the number of POSIX characters actually transferred by an asynchronous I/O operation, and obtaining the error code in case the operation failed, shall correspond to the P1003.4 C-Language Specification operations as shown in Table 6.11.

Table 6.11 – Retrieve Return Status of Asynchronous I/O Operation

C-Language		Ada-Language		Notes
aio_return		Characters_Transferred		
aiocbp	*(I)*	CB	*(I)*	
<return-value>		*<return-value>*		(1)
aio_error		Characters_Transferred		
aiocbp	*(I)*	CB	*(I)*	
<return-value>		*(none)*		(1)

NOTES:

(1) This one Ada operation Characters_Transferred combines the effects of the two P1003.4 C-Language Specificationoperations. In the case that *aio_return* indicates that the operation failed, *aio_error* is called to obtain and set the proper POSIX error code before POSIX_Error is raised. The error code that would be obtained by calling *aio_error* can then be obtained by calling POSIX.Get_Error_Code from the exeption handler for POSIX_Error.

6.7.5.3 Error Handling

In addition to the error conditions specified in the P1003.4 C-Language Specification, Constraint_Error shall be raised if the value of the CB parameter is **null**.

6.7.5.4 References

aio_error, P1003.4 §6.7.5 and *aio_return*, P1003.4 §6.7.6.

6.7.6 Cancel Asynchronous I/O Request

6.7.6.1 Synopsis

```
type Cancelation_Status is
   (Canceled,
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        Not_Canceled,
        Done);
   function Cancel
      (File: in POSIX_IO.File_Descriptor;
       CB:   in Control_Block_Pointer)
     return Cancelation_Status;
   function Cancel
      (File: in POSIX_IO.File_Descriptor)
     return Cancelation_Status;
```

6.7.6.2 Description

The Ada-language operation for cancelling an asynchronous I/O operation shall correspond to the P1003.4 C-Language Specification operation as shown in Table 6.12.

Table 6.12 – Cancel Asynchronous I/O Request

C-Language			Ada-Language	
aio_cancel			`Cancel`	
	fildes	*(I)*	`File`	*(I)*
	aiocbp ($\neq$ NULL)	*(I)*	`CB`	*(I)*
	<*return-value*>		<*return-value*>	
aio_cancel			`Cancel`	
	fildes	*(I)*	`File`	*(I)*
	aiocbp (= NULL)	*(I)*	*(none)*	*(I)*
	<*return-value*>		<*return-value*>	

The Ada-language names for values of the `Cancelation_Status` type shall correspond to the P1003.4 C-Language Specification constants as shown in Table 6.13.

Table 6.13 – Cancelation Status Values

C-Language	Ada-Language
AIO_CANCELED	`Canceled`
AIO_NOTCANCELED	`Not_Canceled`
AIO_ALLDONE	`Done`

6.7.6.3 Error Handling

In addition to the error conditions specified in the P1003.4 C-Language Specification, `Constraint_Error` shall be raised if the value of the `CB` parameter is **`null`**.

6.7.6.4 References

aio_cancel, P1003.4 § 6.7.7.

6.7.7 Wait for Asynchronous I/O Request

6.7.7.1 Synopsis

```
   procedure Await_IO
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
317        (CB:              in out Control_Block_Pointer;
318         Timeout:         in POSIX.Timespec;
319         Masked_Signals:  in POSIX.Signal_Masking := POSIX.RTS_Signals);
320     procedure Await_IO
321        (CB:              in out Control_Block_Pointer;
322         Masked_Signals:  in POSIX.Signal_Masking := POSIX.RTS_Signals);
323     procedure Await_IO
324        (List:            in out Control_Block_List;
325         Timeout:         in POSIX.Timespec;
326         Masked_Signals:  in POSIX.Signal_Masking := POSIX.RTS_Signals);
327     procedure Await_IO
328        (List:            in out Control_Block_List;
329         Masked_Signals:  in POSIX.Signal_Masking := POSIX.RTS_Signals);
```

330 6.7.7.2 Description

331 The Ada-language operation for suspending execution to await the completion of one
332 of a list of asynchronous I/O operations shall correspond to the P1003.4 C-Language
333 Specification operation as shown in Table 6.14.

334 **Table 6.14 – Wait for Asynchronous I/O Request**

335 C-Language		Ada-Language		Notes
336 *aio_suspend*		Await_IO		
337 *list*	*(I)*	CB	*(I)*	
338 *nent* (= 1)	*(I)*	*(none)*	*(I)*	
339 *timeout* (≠ NULL)	*(I)*	Timeout	*(I)*	
340 *(none)*	*(I)*	Masked_Signals	*(I)*	(1)
341 *aio_suspend*		Await_IO		
342 *list*	*(I)*	CB	*(I)*	
343 *nent* (= 1)	*(I)*	*(none)*	*(I)*	
344 *timeout* (= NULL)	*(I)*	*(none)*	*(I)*	
345 *(none)*	*(I)*	Masked_Signals	*(I)*	(1)
346 *aio_suspend*		Await_IO		
347 *list*	*(I)*	List	*(I)*	
348 *nent* (≥ 1)	*(I)*	List'length	*(I)*	
349 *timeout* (≠ NULL)	*(I)*	Timeout	*(I)*	
350 *(none)*		Masked_Signals		(1)
351 *aio_suspend*		Await_IO		
352 *list*	*(I)*	List	*(I)*	
353 *nent* (≥ 1)	*(I)*	List'length	*(I)*	
354 *timeout* (= NULL)	*(I)*	*(none)*	*(I)*	
355 *(none)*	*(I)*	Masked_Signals	*(I)*	(1)

356 NOTES:

357 (1) The Ada parameter Masked_Signals shall specify the set of signals that are to be added
358 to the signal mask that is in effect during the operation. The previous signal mask shall be
359 restored when the operation completes.

360 6.7.7.3 Error Handling

361 In addition to the error conditions specified in the P1003.4 C-Language Specification,
362 Constraint_Error shall be raised if the value of the CB parameter is **null**.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.7.7.4 References

aio_suspend, P1003.4 § 6.7.7.

6.7.8 Asynchronous File Synchronization

6.7.8.1 Synopsis

```
   procedure Synchronize_File(CB: in Control_Block_Pointer);
   procedure Synchronize_Data(CB: in Control_Block_Pointer);
```

6.7.8.2 Description

The Ada-language operations to force asynchronous I/O operations to the *synchronized completion* state shall correspond to the P1003.4 C-Language Specification operation as shown in Table 6.15.

Table 6.15 – Asynchronous File Synchronization

C-Language			Ada-Language	
aio_fsync			`Synchronize_File`	
	op (= O_SYNC)	*(I)*	*(none)*	*(I)*
	aiocbp	*(I)*	`CB`	*(I)*
aio_fsync			`Synchronize_Data`	
	op (= O_DSYNC)	*(I)*	*(none)*	*(I)*
	aiocbp	*(I)*	`CB`	*(I)*

6.7.8.3 Error Handling

In addition to the error conditions specified in the P1003.4 C-Language Specification, `Constraint_Error` shall be raised if the value of the `CB` parameter is **`null`**.

6.7.8.4 References

aio_fsync, P1003.4 § 6.7.9.

6.8 Rationale for Revisions to Input and Output Primitives

6.8.1 Packaging

The new contents of the `POSIX_IO` package are all things which logically belong with the types already declared in that package in P1003.5.

6.8.2 IO Control Blocks

The use of control blocks and buffers is unsafe, since there is a persistent reference to the control block for the duration of the IO operation. This means that if the control block is deallocated and later reallocated for another use, the behavior of

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

the operation is undefined. The same problem occurs with the data buffers that are used for asynchronous I/O. If operation is a write, a dangling reference may result in writing out garbage. If the I/O operation is a read, this may result in overwriting arbitrary memory, which generally leads to chaotic failure of the program. The solution here is to require that the control block and buffer pointer be of types declared in a library level package, so that the storage should persist for the life of the process. If a programmer attempts to use unchecked deallocation on such a buffer, or to use unchecked conversion to obtain a value of the `IO_Buffer_Pointer` type, it becomes the programmer's responsibility to ensure against dangling references.

The idea of providing a deallocation operation that would check that there are no outstanding I/O operations on the block or buffer that is deallocated, was considered. This might be done fairly easily for control blocks, but seems awkward for buffers.

6.8.3 Return Status and Error Status

For asynchronous I/O, the *aio_error* and *aio_return* functions provide completion and error status information that is provided for other POSIX operations via the `POSIX_Error` exception and the `Get_Error_Code` function (parameterless). The `Characters_Transferred` function combines these effects, and is logically complete. (The implementation of this function would need to call *aio_error* to set the error code value when it raises `POSIX_Error` if the operation has not completed normally.) However, using only this function would mean that checking for important normal conditions (e.g. I/O still in progress) would require the raising and handling of an exception. For this reason, the `Get_Error_Code` operation is also provided.

6.8.4 File Status Flags

In P1003.4, O_DSYNC or O_SYNC is passed to *aio_fsync* to specify the kind of synchronization desired. There are constants corresponding to O_DSYNC and O_SYNC declared in `POSIX_Synchronized_IO`, but there they are used as option sets (i.e. flags). However, these constants are not used as flags when they are passed as parameter to the *aio_fsync* operation. Inconsistent type usage has been avoided by splitting this into two operations.

6.8.5 Asynchronous I/O Priority

The `Integer` type is used for the parameter of `Set_Priority` and the return value of `Get_Priority`. This is for the sake of consistency with the type of `System.Priority`, and because it is not clear whether any non-trivial compile-time bounds can be put on the range of allowed priority values.

6.8.6 Arrays

Where the P1003.4 C-Language Specification specifies a pointer to an array and a length as separate in-parameters, they are replaced here by an Ada unconstrained array; the `length` attribute of the array specifies the length. Note that the actual number of POSIX characters transferred is obtainable via the function `Last`.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

In the case of the *aio_suspend* operation it was necessary to split the operation into two forms, so handle the case of a single control block pointer. The C-language interface apparently only specifies a form for an array of pointers, assuming that a single pointer can be treated as if it were an array of length one.

6.8.7 P1003.4 Questions

It is not specified what happens if the *nent* parameter of *aio_suspend* is zero or negative. This issue is avoided in the Ada interface by specifying the index range of `Control_Block_List` to be positive.

6.8.8 Ada 9X Considerations

The object finalization feature of Ada 9X may provide an alternative solution for the problem of dangling references to I/O control blocks and I/O buffers. Using object finalization, it would be possible to automatically cancel all pending I/O operations on exit from the scope where an I/O control block object or I/O buffer is declared. Since the cancelation operation can fail, if the I/O is in progress, the finalization operation would need to repeat the attempt at cancelation until it succeeds. However, it is presently not certain that object finalization will survive in Ada 9X.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

449 *This side is intentionally blank.*

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Section 11: Synchronization

This section describes the services which this standard provides for process synchronization, via counting semaphores.

Specification of Package `POSIX_Semaphores`

```
with POSIX;                              -- from P1003.5
with POSIX_File_Status;                  -- from P1003.5
package POSIX_Semaphores is
   -- §11.1.1 Semaphore Descriptor Type   ...
   -- §11.2.1 Initialize an Unnamed Semaphore   ...
   -- §11.2.2 Destroy an Unnamed Semaphore   ...
   -- §11.2.3 Initialize/ Open a Named Semaphore   ...
   -- §11.2.4 Close a Named Semaphore   ...
   -- §11.2.5 Remove a Named Semaphore   ...
   -- §11.2.6 Lock a Semaphore   ...
   -- §11.2.7 Unlock a Semaphore   ...
   -- §11.2.8 Get the Value of a Semaphore   ...
private
   -- impl.-defined;
end POSIX_Semaphores;
```

11.1 Semaphore Characteristics

11.1.1 Semaphore Descriptor Type

11.1.1.1 Synopsis

```
type Semaphore is private;
```

11.1.1.2 Description

The Ada-language name of the type for designating an open semaphore shall correspond to the P1003.4 C-Language Specification operation as shown in Table 11.1.

Table 11.1 – Semaphore Descriptor Type

C-Language	Ada-Language
sem_t	`Semaphore`

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

11.1.1.3 References

sem_t, P1003.4 § 11.1.

11.2 Semaphore Functions

11.2.1 Initialize an Unnamed Semaphore

11.2.1.1 Synopsis

```
procedure Initialize
   (Sem:       in out Semaphore;
    Is_Shared: in  Boolean;
    Value:     in  Natural);
```

11.2.1.2 Description

The Ada-language operations to initialize an unnamed semaphore shall correspond to the P1003.4 C-Language Specification operation as shown in Table 11.2.

Table 11.2 – Initialize an Unnamed Semaphore

C-Language		Ada-Language	
sem_init		Initialize	
sem	*(I,O)*	Sem	*(I,O)*
pshared	*(I)*	Is_Shared	*(I)*
value	*(I)*	Value	*(I)*

11.2.1.3 References

sem_init, P1003.4 § 11.2.1.

11.2.2 Destroy an Unnamed Semaphore

11.2.2.1 Synopsis

```
procedure Destroy(Sem: in Semaphore);
```

11.2.2.2 Description

The Ada-language operations to destroy an unnamed semaphore shall correspond to the P1003.4 C-Language Specification operation as shown in Table 11.3.

Table 11.3 – Destroy an Unnamed Semaphore

C-Language		Ada-Language	
sem_destroy		Destroy	
sem	*(I,O)*	Sem	*(I,O)*

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

11.2.2.3 References

sem_destroy, P1003.4 § 11.2.2.

11.2.3 Initialize/Open a Named Semaphore

11.2.3.1 Synopsis

```
function Open
   (Name:      in  POSIX.POSIX_String)
   return Semaphore;
function Open_Or_Create
   (Name:      in  POSIX.POSIX_String;
    Value:     in  Natural)
   return Semaphore;
```

11.2.3.2 Description

The Ada-language operations to initialize or open a named semaphore shall correspond to the P1003.4 C-Language Specification operation as shown in Table 11.4.

Table 11.4 – Initialize/Open a Named Semaphore

C-Language		Ada-Language	
sem_open		Open	
sem	*(O)*	<*return-value*>	*(O)*
oflag (= 0)	*(I)*	*(none)*	*(I)*
name	*(I)*	Name	*(I)*
sem_open		Open_Or_Create	
sem	*(O)*	<*return-value*>	*(O)*
oflag (= O_CREAT)	*(I)*	*(none)*	*(I)*
name	*(I)*	Name	*(I)*
value	*(I)*	Value	*(I)*

11.2.3.3 References

sem_open, P1003.4 § 11.2.3.

11.2.4 Close a Named Semaphore

11.2.4.1 Synopsis

```
procedure Close(Sem: in Semaphore);
```

11.2.4.2 Description

The Ada-language operation to close a named semaphore shall correspond to the P1003.4 C-Language Specification operation as shown in Table 11.5.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 11.5 – Close a Named Semaphore

C-Language	Ada-Language
sem_close *sem (I)*	Close Sem (I)

11.2.4.3 References

sem_close, P1003.4 § 11.2.4.

11.2.5 Remove a Named Semaphore

11.2.5.1 Synopsis

```
procedure Unlink(Name: in POSIX.POSIX_String);
```

11.2.5.2 Description

The Ada-language operation to remove a named semaphore shall correspond to the P1003.4 C-Language Specification operation as shown in Table 11.6.

Table 11.6 – Remove a Named Semaphore

C-Language	Ada-Language
sem_unlink *name (I)*	Unlink Name (I)

11.2.5.3 References

sem_unlink, P1003.4 § 11.2.5.

11.2.6 Lock a Semaphore

11.2.6.1 Synopsis

```
procedure Wait
   (Sem:            in Semaphore;
    Masked_Signals: in POSIX.Signal_Masking:= POSIX.No_Signals);
function Try_Wait
   (Sem:    Semaphore) return Boolean;
```

11.2.6.2 Description

The Ada-language operations to lock a semaphore shall correspond to the P1003.4 C-Language Specification operations as shown in Table 11.7.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 11.7 – Lock a Semaphore

C-Language			Ada-Language			Notes
sem_wait			`Wait`			
	sem	*(I)*		`Sem`	*(I)*	
	(none)	*(I)*		`Masked_Signals`	*(I)*	(1)
sem_trywait			`Try_Wait`			
	sem	*(I)*		`Sem`	*(I)*	
	<return-value>			*<return-value>*		(2)

NOTES:

(1) The Ada parameter `Masked_Signals` specifies the set of signals that are to be added to the signal mask that is in effect during the operation. The previous signal mask shall be restored when the operation completes.

(2) The Ada function returns `False` if and only if the P1003.4 C-Language Specification operation would return a value of -1 with error code [EAGAIN].

11.2.6.3 Error Handling

The error handling is as specified in the P1003.4 C-Language Specification, except that no exception is raised for the P1003.4 C-Language Specification "error" code [EAGAIN], since in this case the function `Try_Wait` returns false.

11.2.6.4 References

sem_wait, P1003.4 § 11.2.6; *sem_trywait*, P1003.4 § 11.2.6.

11.2.7 Unlock a Semaphore

11.2.7.1 Synopsis

```
procedure Post(Sem: in Semaphore);
```

11.2.7.2 Description

The Ada-language operation to unlock a semaphore shall correspond to the P1003.4 C-Language Specification operation as shown in Table 11.8.

Table 11.8 – Unlock a Semaphore

C-Language			Ada-Language		
sem_post			`Post`		
	sem	*(I)*		`Sem`	*(I)*

11.2.7.3 References

sem_post, P1003.4 § 11.2.7.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

11.2.8 Get the Value of a Semaphore

11.2.8.1 Synopsis

```
function Get_Value(Sem: in Semaphore) return Integer;
```

11.2.8.2 Description

The Ada-language operation to obtain the value of a semaphore shall correspond to the P1003.4 C-Language Specification operation as shown in Table 11.9.

Table 11.9 – Get the Value of a Semaphore

C-Language	Ada-Language
sem_get_value *sem* (I) <*return-value*>	Get_Value Sem (I) <*return-value*>

11.2.8.3 References

sem_get_value, P1003.4 § 11.2.8

11.3 Rationale

Semaphores are an important tool for synchronization between *processes*. They do not overlap in functionality with rendezvous, which is intended for synchronization between Ada tasks. (P1003.4 semaphores may possibly also be of some use between tasks, for implementations where P1003.4a threads are supported and used to implement Ada tasks, but such usage will not be portable.)

11.3.1 Semaphore Creation

The *sem_init* operation has been split into two operations, according to the value of the *oflag* parameter; this is for consistency with the treatment of *open* in P1003.5. For the same reason, the out-parameter is changed to a function return value.

Apparently, the P1003.4 *sem_init* and *sem_open* operations differ in whether the semaphore is assumed to be shared. The *sem_init* operation has a parameter pshared to tell whether the semaphore is shared. The *sem_open* operation implicitly creates a shared semaphore.

11.3.2 Semaphore Locking

The need for raising an exception corresponding to [EAGAIN] was eliminated by adding a Boolean return value to `Try_Wait`.

The `Try_Wait` operation has no `Masked_Signals` parameter, since (unlike `Wait`) it cannot block.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

11.3.3 Semaphore Values

The P1003.4 document intentionally leaves unspecified the value that is returned by *sem_get_value* in the case that the semaphore is locked; the value returned may be zero (i.e. the actual value of the semaphore) or it may be a negative number whose absolute value is the count of pending lock operations on the semaphore. For this reason, the value returned by `Get_Value` is of type `Integer`. On the other hand, the value of the semaphore itself is always non-negative. Therefore, the operations *sem_init* and *sem_open* do not allow the value of a semaphore to be set to a negative value, and the `Value` parameter of the operations `Initialize` and `Open_Or_Create` is of subtype `Natural`.

11.3.4 P1003.4a Considerations

This section defines operations that can block an entire process. As with all such operations, P1003.4a specifies that they need only block the calling thread (but may still block the whole process). As with all blocking system calls, this results in a potential change of semantics when moving between a POSIX implementation that does not support threads and one that does support threads. In particular, using semaphores to synchronize between Ada tasks within a single process may be workable if POSIX threads are supported and Ada tasks map to POSIX threads, but otherwise it is likely to cause deadlock within the process.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

204 *This side is intentionally blank.*

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Section 12: Memory Management

This section specifies interfaces for memory management, including locking portions of a process's address space so that they are continually resident in memory, and mapping files and shared memory objects to the address space of a process.

12.1 Memory Locking Functions

Specification of Package POSIX_Memory_Locking

```
with POSIX;                              -- from P1003.5
package POSIX_Memory_Locking is
   -- §12.1.1 Lock/ Unlock a Process's Address Space   ...
end POSIX_Memory_Locking;
```

Specification of Package POSIX_Memory_Range_Locking

```
with System;
with POSIX;                              -- from P1003.5
with POSIX_Address_Arithmetic;
package POSIX_Memory_Range_Locking is
   -- §12.1.2 Lock/ Unlock a Range of Process Address Space   ...
end POSIX_Memory_Range_Locking;
```

12.1.1 Lock/Unlock a Process's Address Space

12.1.1.1 Synopsis

```
   type Memory_Locking_Options is new POSIX.Option_Set;
   Current_Pages: constant Memory_Locking_Options:= impl.-defined;
   Future_Pages:  constant Memory_Locking_Options:= impl.-defined;
   procedure Lock_All(Options: in Memory_Locking_Options);
   procedure Unlock_All;
```

12.1.1.2 Description

The Ada-language names for data types associated with process memory locking shall correspond to data types of the P1003.4 C-Language Specification as shown in Table 12.1.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 12.1 – Memory Locking Types

C-Language	Ada-Language
int	`Memory_Locking_Options`

`Memory_Locking_Options` is used to specify whether the current pages, the future pages, or both the current and future pages of the process's memory are to be locked.

The Ada-language names for constants of the type `Memory_Locking_Options` shall correspond to constants of the P1003.4 C-Language Specification as shown in Table 12.2.

Table 12.2 – Memory Locking Option Constants

C-Language	Ada-Language
MCL_CURRENT	`Current_Pages`
MCL_FUTURE	`Future_Pages`

The Ada-language operations for locking and unlocking all of a process's address space shall correspond to the P1003.4 C-Language Specification operations as shown in Table 12.3.

Table 12.3 – Lock/Unlock a Process's Address Space

C-Language	Ada-Language
mlockall *flags* *(I)*	`Lock_All` `Options` *(I)*
munlockall	`Unlock_All`

12.1.1.3 References

MCL_CURRENT, P1003.4 § 12.1.1; MCL_FUTURE, P1003.4 § 12.1.1; *mlockall*, P1003.4 § 12.1.1; *munlockall*, P1003.4 § 12.1.1.

12.1.2 Lock/Unlock a Range of Process Address Space

12.1.2.1 Synopsis

```
procedure Lock_Range
   (First:  in System.Address;
    Length: in POSIX_Address_Arithmetic.Address_Offset);
procedure Unlock_Range
   (First:  in System.Address;
    Length: in POSIX_Address_Arithmetic.Address_Offset);
```

12.1.2.2 Description

The Ada-language operations for locking and unlocking specific regions of a process's address space shall correspond to the P1003.4 C-Language Specification operations as shown in Table 12.4.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 12.4 – Lock/Unlock a Range of a Process's Address Space

C-Language			Ada-Language	
mlock			Lock_Range	
	addr	*(I,O)*	First	*(I,O)*
	len	*(I)*	Length	*(I)*
munlock			Unlock_Range	
	addr	*(I,O)*	First	*(I,O)*
	len	*(I)*	Length	*(I)*

12.1.2.3 References

mlock, P1003.4 § 12.1.2; *munlock*, P1003.4 § 12.1.2; Align_To_Page, Length§ 2.6.1.

12.2 Memory Mapping Functions

Specification of Package POSIX_Memory_Mapping

```
with POSIX;                              -- from P1003.5
with POSIX_IO;                           -- from P1003.5
with POSIX_Signals;                      -- from P1003.5
with SYSTEM;
with POSIX_Address_Arithmetic;
package POSIX_Memory_Mapping is
   -- § 12.2.1 Map Process Addresses to a Memory Object   ...
   -- § 12.2.2 Memory Unmapping   ...
   -- § 12.2.3 Change Memory Protection   ...
   -- § 12.2.4 Memory Object Synchronization   ...
end POSIX_Memory_Mapping;
```

12.2.1 Map Process Addresses to a Memory Object

12.2.1.1 Synopsis

```
   type Protection_Options is new POSIX.Option_Set;
   Allow_Read:    constant Protection_Options:= impl.-defined;
   Allow_Write:   constant Protection_Options:= impl.-defined;
   Allow_Execute: constant Protection_Options:= impl.-defined;
   Allow_Nothing: constant Protection_Options:= impl.-defined;
   function Map_Memory
      (First:          in  System.Address;
       Length:         in  POSIX_Address_Arithmetic.Address_Offset;
       Protection:     in  Protection_Options;
       Share_Changes:  in  Boolean;
       Exact_Address:  in  Boolean;
       File:           in  POSIX_IO.File_Descriptor;
       Offset:         in  POSIX.IO_Count)
     return System.Address;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12.2.1.2 Description

The Ada-language type used in memory mapping shall correspond to the type used in the P1003.4 C-Language Specification as shown in Table 12.5.

Table 12.5 – Memory Mapping Types

C-Language	Ada-Language
int	`Protection_Options`

The `Protection_Options` type is used to specify the form of access that is requested for a mapping. The Ada-language constants of the `Protection_Options` type shall correspond to constants in the P1003.4 C-Language Specification as shown in Table 12.6.

Table 12.6 – Memory Protection Constants

C-Language	Ada-Language
PROT_READ	`Allow_Read`
PROT_WRITE	`Allow_Write`
PROT_EXEC	`Allow_Execute`
PROT_NONE	`Allow_Nothing`

The Ada-language constant `POSIX_Address_Arithmetic.Null_Address` shall correspond to the address value zero as shown in Table 12.7.

Table 12.7 – Zero Address Constant

C-Language	Ada-Language	Notes
0	`Null_Address`	(1)

NOTES:

(1) `POSIX_Address_Arithmetic.Null_Address` corresponds to the address value zero, in the context where it is used in the P1003.4 C-Language Specification to specify that there is no preference for the address at which mapping should take place.

The Ada-language operation for memory mapping shall correspond to the P1003.4 C-Language Specification operation as shown in Table 12.8.

12.2.1.3 References

mmap, P1003.4 § 12.2.1; `Align_To_Page`, `Length` § 2.6.1.

12.2.2 Memory Unmapping

12.2.2.1 Synopsis

```
procedure Unmap_Memory
   (First:  in System.Address;
    Length: in POSIX_Address_Arithmetic.Address_Offset);
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 12.8 – Map Process Addresses to a Memory Object

C-Language			Ada-Language		Notes
mmap			Map_Memory		
	addr	*(I)*	First	*(I)*	(1)
	len	*(I)*	Length	*(I)*	
	prot	*(I)*	Protection	*(I)*	
	flags (&& MAP_SHARED)	*(I)*	Share_Changes	*(I)*	
	flags (&& MAP_FIXED)	*(I)*	Exact_Address	*(I)*	
	fildes	*(I)*	File	*(I)*	
	off	*(I)*	Offset	*(I)*	
	<*return-value*>		<*return-value*>		

NOTES:

(1) The effect of address value zero in the P1003.4 C-Language Specification is obtained in Ada by passing `Null_Address` as the actual parameter.

12.2.2.2 Description

The Ada-language operation for undoing memory mapping shall correspond to the P1003.4 C-Language Specification operation as shown in Table 12.9.

Table 12.9 – Unmap Previously Mapped Addresses

C-Language		Ada-Language	
munmap		Unmap_Memory	
addr	*(I)*	First	*(I)*
len	*(I)*	Length	*(I)*

12.2.2.3 References

munmap, P1003.4 § 12.2.2.

12.2.3 Change Memory Protection

12.2.3.1 Synopsis

```
procedure Change_Protection
   (First:      in System.Address;
    Length:     in POSIX_Address_Arithmetic.Address_Offset;
    Protection: in Protection_Options);
```

12.2.3.2 Description

The Ada-language operation for changing memory protection shall correspond to the P1003.4 C-Language Specification operation as shown in Table 12.10.

12.2.3.3 References

mprotect, P1003.4 § 12.2.3.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 12.10 – Change Memory Protection

C-Language		Ada-Language	
mprotect		Change_Protection	
addr	*(I)*	First	*(I)*
len	*(I)*	Length	*(I)*
prot	*(I)*	Protection	*(I)*

12.2.4 Memory Object Synchronization

12.2.4.1 Synopsis

```
procedure Synchronize_Memory
      (First:                  in System.Address;
       Length:                 in POSIX_Address_Arithmetic.Address_Offset;
       Wait_For_Completion:    in Boolean:= True;
       Invalidate_Cached_Data: in Boolean:= False);
```

12.2.4.2 Description

The Ada-language type for specifying memory synchronization options shall correspond to types in the P1003.4 C-Language Specification as shown in Table 12.11.

Table 12.11 – Memory Mapping Types

C-Language	Ada-Language
int	Synchronization_Options

The Ada-language operations for sychronizing the state of mapped memory or writing the content of mapped memory to permanent storage shall correspond to the operation in the P1003.4 C-Language Specification as shown in Table 12.12.

Table 12.12 – Memory Object Synchronization

C-Language		Ada-Language		Notes
msync		Synchronize_Memory		
addr	*(I)*	First	*(I)*	
len	*(I)*	Length	*(I)*	
flags (&& MS_SYNC)	*(I)*	Wait_For_Completion	*(I)*	(1)
flags (&& MS_INVALIDATE)	*(I)*	Invalidate_Cached_Data	*(I)*	

NOTES:

(1) The value `True` shall correspond to the flag MS_SYNC being set in *flags*, and the value `False` shall correspond to the flag MS_ASYNC being set. P1003.4 specifies that either MS_SYNC or MS_ASYNC shall be specified, but not both.

12.2.4.3 References

MS_ASYNC, MS_SYNC, MS_INVALIDATE, P1003.4 § 12.2.4; *msync*, P1003.4 § 12.2.4.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12.3 Shared Memory Functions

Specification of Package POSIX_Shared_Memory_Objects

```
with POSIX;                          -- from P1003.5
with POSIX_IO;                       -- from P1003.5
with POSIX_Permissions;              -- from P1003.5
package POSIX_Shared_Memory_Objects is
   -- §12.3.1 Open a Shared Memory Object  ...
   -- §12.3.2 Remove a Shared Memory Object  ...
end POSIX_Shared_Memory_Objects;
```

12.3.1 Open a Shared Memory Object

12.3.1.1 Synopsis

```
   function Open_Shared_Memory
      (Name:           POSIX.POSIX_String;
       Mode:           POSIX_IO.File_Mode;
       Options:        POSIX_IO.Open_Option_Set;
       Masked_Signals: POSIX.Signal_Masking := POSIX.RTS_Signals)
     return POSIX_IO.File_Descriptor;
   function Open_Or_Create_Shared_Memory
      (Name:           POSIX.POSIX_String;
       Mode:           POSIX_IO.File_Mode;
       Permissions:    POSIX_Permissions.Permission_Set;
       Options:        POSIX_IO.Open_Option_Set;
       Masked_Signals: POSIX.Signal_Masking := POSIX.RTS_Signals)
     return POSIX_IO.File_Descriptor;
```

12.3.1.2 Description

The Ada-language operations for opening or creating a shared memory object shall correspond to the operation in the P1003.4 C-Language Specification as shown in Table 12.13.

Note that the POSIX_IO.Close operation, which is specified in P1003.5 is used to close a shared memory object.

12.3.1.3 References

shm_open, P1003.4 §12.3.1; *ftruncate*, P1003.4 §5.6.7; *close*, P1003.4 §6.3.1.

12.3.2 Remove a Shared Memory Object

12.3.2.1 Synopsis

```
   procedure Unlink_Shared_Memory
      (Name: in POSIX.POSIX_String);
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 12.13 – Open a Shared Memory Object

C-Language			Ada-Language			Notes
shm_open			`Open_Shared_Memory`			
	path	*(I)*		`Name`	*(I)*	
	oflag	*(I)*		`Mode`	*(I)*	(1)
	oflag	*(I)*		`Options`	*(I)*	(2)
	(none)	*(I)*		`Masked_Signals`	*(I)*	(4)
	<return-value>			*<return-value>*		
shm_open			`Open_Or_Create_Shared_Memory`			
	path	*(I)*		`Name`	*(I)*	
	oflag	*(I)*		`Mode`	*(I)*	(1)
	oflag	*(I)*		`Options`	*(I)*	(3)
	mode	*(I)*		`Permissions`	*(I)*	
	(none)	*(I)*		`Masked_Signals`	*(I)*	(4)
	<return-value>			*<return-value>*		

NOTES:

(1) Only `Read_Only` and `Read_Write` are permitted. `Read_Only` shall correspond to O_RDONLY. `Read_Write` shall correspond to O_RDWR.

(2) `Options` shall correspond to the O_TRUNC option in *oflag*.

(3) `Options` shall correspond to the O_TRUNC and O_EXCL options in *oflag*.

(4) The Ada parameter `Masked_Signals` shall specify the set of signals that are to be added to the signal mask that is in effect during the operation. The previous signal mask shall be restored when the operation completes.

12.3.2.2 Description

The Ada-language operation for deleting a shared memory object shall correspond to the operation in the P1003.4 C-Language Specification as shown in Table 12.14.

Table 12.14 – Remove a Shared Memory Object

C-Language			Ada-Language		
shm_unlink			`Unlink_Shared_Memory`		
	path	*(I)*		`Name`	*(I)*

12.3.2.3 References

shm_unlink, P1003.4 § 12.3.2.

12.4 Rationale for Memory Management

12.4.1 Rationale for Process Memory Locking

This is a very direct binding. In the case of the operations on the whole address space, such a direct binding seems safe and appropriate for use by an Ada application programmer. In the case of the range locking operations, it does not seem appropriate.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12.4.2 Range Locking

Because it is perceived as riskier to use, range locking is in a package separate from the global locking and unlocking operations.

The locking operations on ranges seem awkward to use in a portable fashion. For locking, an Ada application programmer will have difficulty determining the exact correspondences between memory ranges and Ada entities. It is well known that the `address` and `size` attributes are not portable, especially in the presence of so-called dope (implicitly allocated record and array components used by the Ada implementation for address computations and constraint checking). It is also the case that the values of certain objects may reside only in registers some or all of the time; thus, it may not be meaningful to speak of the address of some objects.

For unlocking, there also seems to be a problem determining whether two objects occupy the same page of memory, so as to safely unlock the space of one without accidentally also unlocking the space of another. (These problems are similar to those with memory mapping, which is discussed in § 12.4.3.)

As an alternative to the present range locking interface, or in addition to it, the possibility of providing safer interfaces should be considered.

One alternative is a pragma, that would specify to the Ada runtime system what locking needs to be done, for a specific code entity (e.g. an interrupt handler) or data structure (e.g. a buffer). For example, consider

`pragma Lock_In_Memory(` *name_list* `);`.

Using the information provided by such a pragma, the compiler could generate the necessary sytem calls to lock the specified objects into memory, and no explicit calls by the application are necessary. Problems with this solution include: (1) it might be used for local objects of procedures (on the stack), without matching unlocking on procedure return; (2) it might be used on heap objects, without matching unlocking; (3) it requires special support in the compiler.

Another alternative to using explicit addresses in the interface is to use pointers. For example, consider:

```
generic
  type Object_Type is limited private;
  type Access_Type is access Object_Type;
procedure Lock_In_Memory(Pointer: in Access_Type);
```

If `Ptr` is an access value, a call to an appropriate instantiation of `Lock_In_Memory` could be used to lock the pages containing the object designated by `Ptr` into memory. Portable use of such an interface would be limited to heap-allocated data, with the concomitant problems of safe unlocking. To use this interface with statically allocated data would require unchecked conversion from address to an access value, or some similar non-portable trick.

Unfortunately, this approach does not seem workable for code. In theory, it might be

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

used for tasks created via allocators, but in practice it is likely that a task access value may not be sufficient for the implementation to determine the range of addresses occupied by the code of the corresponding task.

Range locking/unlocking operations based on addresses would still appear to be necessary, even if the higher-level interfaces described above were provided, because the higher-level interfaces do not provide the full functionality of the P1003.4 memory locking. This situation is analogous to the situation in P1003.5, where `Start_Process` is provided to allow a programmer to avoid the dangers of `Fork` and `Exec`, but raw `Fork` and `Exec` are also provided.

12.4.3 Problems with Memory Mapping

The present interface for memory mapping, which is a very direct binding to P1003.4, is likely to be an "unsafe" one for Ada users.

A big problem with the use of shared memory is the danger of undesired compiler optimizations, namely the possibility that the compiler may assume that the value of a variable can only be modified by code generated by that compiler. Based on this assumption and data flow analysis, the compiler may omit loads and stores to memory, keeping the value of the variable in a register. It may also delete entirely any code that computes values that appear to be "dead" – where the value computed is assigned to a variable that is overwritten before being read again. This kind of optimization is incorrect if the variable is in shared memory.

To prevent such undesired (and harmful) optimizations on objects that are in shared memory, some way is needed to inform the Ada compiler that these objects may be read and updated concurrently by other processes. The pragmas `volatile` and `atomic`, which are included in the Ada 9X language proposals, provide this capability. The possibility of introducing such pragmas in this POSIX standard was considered, but was rejected on the grounds that this standard should not overlap with the domain of the Ada 9X language standard. In Ada 83, lacking these pragmas, several work-arounds are known. One that may work on some implementations is to use access values to denote the objects in shared memory; this may reduce the likelihood of data-flow analysis detecting candidates for such undesired optimizations. The other alternative is to use implementation-defined pragmas or compilation flags to suppress optimization.

Another problem is uncertainty about what the Ada runtime system may be already doing with shared memory; for example, if the runtime system is using multiple POSIX processes to implement a single Ada program — as it may do to reduce the blocking effect of certain system calls, or to take advantage of a multiprocessor — there is danger of the application interfering with the Ada runtime system. Also, there is uncertainty about the exact memory layout used by the Ada language implementation; this problem also arises with memory range locking (see § 12.4.2). Ada compilers are not very uniform in the way they support the type `System.Address` and address clauses. In particular, some compilers do not support arithmetic on addresses, or allow non-static expressions in address clauses. There is uncertainty about whether the compiler will generate code that reads and writes data through

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

to memory. In the absence of pragmas specifying otherwise (e.g. the proposed Ada 9X `volatile`), there is no guarantee that a reference or assignment to a shared memory object in the Ada source code will result in a compiler generating actual read or write operations to memory. Since the compiler will not know that a given range of addresses is in shared memory, it may choose to retain values in registers, or to suppress what appear to be redundant load and store operations.

To make good use of the memory-mapping and memory-region locking capabilities, an application needs to be able to:

— Do arithmetic on addresses. (See 2.6.)

— Convert addresses to usable access-type values.

— Specify the address of a data object, via an address clause with a non-static expression.

— Insert system calls, to set open files and shared memory objects and establish mappings, before the elaboration of the Ada objects which are to occupy the mapped address ranges – including objects declared in library packages.

— Suppress optimizations that are not correct in the presence of concurrent access to data from another process.

A standard real-time Ada-language binding should impose uniformity conditions on compliant compilers, so that these capabilities are available in a portable form.

12.4.4 `Null_Address`

The constant `POSIX_Address_Arithmetic.Null_Address` is provided because of the special case treatment of the address value zero by the `Map_Memory` operation. This is required since the Ada type `System.Address` is not required to be an integer type. The name `POSIX_Address_Arithmetic.Null_Address` follows the recommendation of UI-0043 of the ISO/IEC JTC1 SC22 WG9 Uniformity Rapporteur Group (see § 2.6).

12.4.5 Higher Level Interfaces

It seems desirable to have a higher level interface for memory mapping, which would not require the user to specify the address and size of mapped objects. There are at least two mechanisms for higher level interfaces which seem worth considering:

— Pragmas.

— Generic allocators.

These ideas are elaborated further below. However, it will be seen that neither of them completely covers the full range of situations where memory mapping may be needed, and so they cannot entirely replace the address-based interface.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12.4.5.1 Pragmas.

The main advantage of pragmas is that they allow a broader range of parameters. In particular, it is possible to specify a list of package names as parameters to a pragma. This seems desirable. If a set of packages could be mapped to shared memory, processes could communicate in a natural way through the shared data objects in those packages. There is no way to do this with a generic unit or subprogram call.

A pragma could specify to the compiler that certain named data objects or packages should be mapped to shared memory or a file, specify the shared memory object or file to which they should be mapped, and specify the desired protection. The open or create operation could be separate, or a shared memory object could be created automatically, if not already present. The effect of the pragma would be to cause the declared objects of the packages to be mapped to the specified file or shared memory object, sequentially, starting with the beginning of the file. If several processes each apply this pragma to the same list of packages, the data storage of the packages would be overlayed exactly.

The main reason such a pragma has not been included in the present draft interface is that it would require special compiler support. For the most part, the P1003.5 interface has avoided specifying interfaces that would require changes to Ada compilers.

There is also some tricky issues, including how to define the range over which the effect of the pragma persists, how to handle runtime errors arising from the mapping or unmapping operations, and how to provide for synchronization. System calls would be required on entry and exit to the scope over which the pragma has its effect. These system calls could raise errors. Suppose, for example, that one wishes to map the data of a library-level package to shared memory, and that this is specified by a pragma in the specification of that package. In order to guarantee that the mapping is in place before the data can be accessed, the system call to set up the mapping must be done at the beginning of elaboration of the package specification. However, if this raises an exception there is no way for the user to catch the exception and handle it. It seems to follow that the Ada runtime system must handle the error, if any, and then kill the program. This seems rather inflexible, but it also seems to be an inherent problem with Ada's lack of provision for user run-time control over the elaboration process.

An attempt at a compromise solution might be possible, by introducing a new abstraction. Let's call this a *mapped memory region*. A user would declare one or more shared memory regions, in packages. The system calls to set up mapping on these regions would be in the package bodies. Program units that depend on these packages could then use pragmas to specify a binding between various packages and objects with an open shared memory region. The mapped memory regions would be automatically closed on program exit. The synchronization operations would apply to an entire shared memory region. The same interface could be used for locking regions of the process' address space into real memory.

For example, consider the following possible package interface:

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
441 with POSIX;                          -- from P1003.5
442 with POSIX_IO;                       -- from P1003.5
443 with POSIX_Signals;                  -- from P1003.5
444 with SYSTEM;
445 package POSIX_Memory_Mapping is
446    type Mapped_Memory_Region is limited private;
447    type Protection_Options is new POSIX.Option_Set;
448    Allow_Read:    constant Protection_Options:= impl.-defined;
449    Allow_Write:   constant Protection_Options:= impl.-defined;
450    Allow_Execute: constant Protection_Options:= impl.-defined;
451    Allow_Nothing: constant Protection_Options:= impl.-defined;
452    procedure Open_Mapped_Memory_Region
453       (Region:           in out Mapped_Memory_Region;
454        Protection:       in Protection_Options;
455        Share_Changes:    in Boolean;
456        Lock_in_Memory:   in Boolean;
457        Name:             in POSIX.POSIX_String;
458        Mode:             POSIX_IO.File_Mode;
459        Options:          POSIX_IO.Open_Option_Set;
460        Is_Shared_Memory: in Boolean);
461    procedure Change_Protection
462       (Region:     in out Mapped_Memory_Region;
463        Protection: in Protection_Options);
464    procedure Synchronize_Memory
465       (Region: in out Mapped_Memory_Region;
466        Wait_For_Completion: in Boolean:= True;
467        Invalidate_Cached_Data: in Boolean:= False);
468 private
469    -- impl.-defined;
470 end POSIX_Memory_Mapping;
```

471 `Open_Mapped_Memory_Region` has the following effects:

472 1. Verify that *mmap* is supported.

473 2. Create a normal storage object to save the parameter values.

474 3. Check that the `Protection` and `Share_Changes` parameters are consistent
475 with the `Mode` and `Options` parameters, as specified for *mmap*.

476 4. If `Is_Shared_Memory` is `True`, check that the `Mode` and `Options` parameters
477 are consistent with *shm_open*.

478 5. If `Allow_Write` is specified, verify that memory protection is supported.

479 6. If `Share_Changes` is `False`, verify that this option is supported.

480 7. If `Lock_In_Memory` is `True`, verify that memory locking is supported.

481 8. Call `Open` or `Open_Shared_Memory`, as specified by the parameter `Is_Shared_-`
482 `Memory`, to open the named object in the specified mode.

483 The implementation automatically closes the file or shared memory object, undoing
484 the mapping, on exit from the scope of the mapped memory region object. The effect
485 of unchecked deallocation on an open mapped memory region object is undefined.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The error handling is as for Open and Open_Shared_Memory, plus the following:

1. Program_Error shall be raised if the mapped memory region is already open.
2. POSIX_Error shall be raised, and the error code shall be as specified, for the following conditions:

   Permission_Denied
   : The region was not opened for read.

     The region was not opened for write, and Allow_Write was specified for a Share_Changes mapping.

   Resource_Temporarily_Unavailable
   : There are insufficient memory resources to reserve for locking pages into memory, if required by *mlockall*.

   Invalid_Argument
   : Allow_Write is specified and the {_POSIX_MEMORY_PROTECTION} option is not supported.

   No_Such_Operation_On_Device
   : Name refers to an object for which *mmap* is meaningless, such as a terminal.

   Operation_Not_Implemented
   : *mmap* is not supported by this implementation. Lock_In_Memory is True and *mlock* is not supported by this implementation.

   Operation_Not_Permitted
   : Lock_In_Memory was True and the calling process does not have the appropriate privilege to perform the requested operation.

   Operation_Not_Supported
   : Share_Changes was False and the implementation does not support this functionality.

Synchronize_Memory has the effect of *msync* on the entire region of memory currently mapped by the specified Mapped_Memory_Region object.

The error conditions are:

Resource_Busy
: The mapped memory region was opened with Lock_In_Memory equal to True and Invalidate_Cached_Data is True.

Operation_Not_Implemented
: *msync* is not supported by this implementation.

Change_Protection has the effect of changing the protection of the entire region of memory currently mapped by the specified Mapped_Memory_Region object.

The error conditions are:

Permission_Denied
: The region was not opened for read.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

525 The region was not opened for write, and `Allow_Write` is specified for a
526 `Share_Changes` mapping.

527 `Operation_Not_Implemented`
528 *mprotect* is not supported by this implementation.

529 `Resource_Temporarily_Unavailable`
530 `Allow_Write` is specified, `Share_Changes` is `False`, and there are insuffi-
531 cient memory resources to reserve for locking the private pages, if required.

532 The following pragma is used to specify that all the declared data objects and access
533 collections of a package are to be mapped to a specified mapped memory region.

534 `pragma Map_Memory(` *mapped_memory_region_name, package_name);*

535 This pragma must appear in the package specification. The effect is to cause the
536 memory occupied by the package to be allocated within the specified mapped memory
537 region, during the elaboration of the package.

538 The elaboration of the package raises `Program_Error` if the named mapped memory
539 region is not open.

540 Mapping may fail at any point during the elaboration of the specification or body of
541 a mapped package, and `POSIX_Error` will be raised with error code `Not_Enough_-`
542 `Space`, for any of the following conditions:

543 — The mapping could not be locked in memory, if required by `Lock_All`, or `Lock_-`
544 `In_Memory`, due to a lack of resources.

545 — There is insufficient room in the address space, or the mapped file, to effect the
546 mapping.

547 A major weakness of this proposal is that, to be useful across processes, it is necessary
548 to be able to predict that the elaboration order of the mapped packages will be the
549 same for each of the processes – so that corresponding Ada objects map to the same
550 addresses in the shared memory object or offsets in the mapped file.

551 It is certainly safe to use this interface, so long as only one package is mapped to
552 each file or shared memory object. It may also be possible in some cases to use **with**
553 clauses and the pragma `Elaborate` to obtain a consistent elaboration order for a set
554 of packages that are mapped to the same object.

555 ### 12.4.5.2 Generic allocators.

556 A fairly simple extension to the present interface could provide for convenient but
557 rudimentary access to shared memory, without using addresses. A generic operation
558 could be defined that would return an access value of a given type that points to the
559 address of a shared memory object or memory mapped file. This operation would
560 behave somewhat like an allocator. It would take an access type as a generic formal
561 type, and the instantiation take the name of a shared memory object or file as a
562 parameter.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
generic
  type Designated_Type is private;
  type Access_Type is access Designated_Type;
function Get_Mapped_Memory_Access
      (Protection:    Protection_Options;
       Share_Changes: Boolean;
       File:          POSIX_IO.File_Descriptor;
       Offset:        POSIX.IO_Count) return Access_Type;
```

The interface could be further simplified, by allowing only one Ada object to be mapped to each file or shared memory object. In that case, the `Offset` parameter could be dropped.

Further simplifications could combine the `Open` or `Open_Shared_Memory` operation with the mapping, in a fashion similar to the way `Start_Process` combines `Fork` and `Exec` in P1003.5.

12.4.6 Shared Memory Objects

The present interface for shared memory objects is based on the P1003.5 treatment of `Open` and `Open_Or_Create`.

12.4.7 P1003.4 Questions

12.4.8 Ada 9X Considerations

In the Ada 9X mapping specification [8] there are standard integer types that would seem appropriate for expressing memory range sizes; these are `System.Storage_Count` and `System.Storage_Offset`.

In [8] it is possible to define access types to procedures and functions as well as data and tasks, and it is possible to obtain access values that designate objects that are not created by allocators. Thus, it seems some of the limitations of the approach suggested above for specifying range locking using access types, may be lifted. However, the limitations on subprogram access types and generic formal subprograms are sufficiently tight that this mechanism is unlikely to be of use for locking the code of subprograms.

Note also that the access parameter mechanism being proposed for Ada 9X would allow the generic memory mapping interface above to be simplified:

```
 generic
  type Object_Type is limited private;
procedure Lock_In_Memory(Access: access Object_Type);
```

Ada 9X will apparently provide pragmas to suppress compiler optimizations that would interfere with shared memory use.

If the concept of program partitions survives in Ada 9X, it may be useful to use passive partitions as logical collections for memory-mapping to a single file or shared memory object.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Section 13: Execution Scheduling

This section describes the scheduling facilities available under the standard.

Specification of Package `POSIX_Priority_Scheduling`

```
with POSIX_Process_Identification;
with POSIX;
package POSIX_Priority_Scheduling is
   -- §13.1.1 Scheduling Parameters ...
   -- §13.2.1 Scheduling Policies ...
   -- §13.3.1 Set Scheduling Parameters ...
   -- §13.3.2 Get Scheduling Parameters ...
   -- §13.3.3 Set Scheduling Policy and Scheduling Parameters ...
   -- §13.3.4 Get Scheduling Policy ...
   -- §13.3.5 Yield CPU ...
   -- §13.3.6 Get Scheduling Parameter Limits ...
private
   -- impl.-defined;
end POSIX_Priority_Scheduling;
```

13.1 Scheduling Parameters

13.1.1 Scheduling Parameters

13.1.1.1 Synopsis

```
   subtype Scheduling_Priority is Integer;
   type Scheduling_Parameters is private;
   function Get_Priority(Parameters: Scheduling_Parameters)
     return Scheduling_Priority;
   procedure Set_Priority
     (Parameters: in out Scheduling_Parameters;
      Priority:   in Scheduling_Priority);
```

13.1.1.2 Description

The Ada-language names for data types associated with scheduling parameters shall correspond to data types of the P1003.4 C-Language Specification as shown in Table 13.1.

The Ada-language names for read and write operations of the `Sheduling_Parameters` data type shall correspond to components of the *sched_param* type of the P1003.4 C-Language Specification as shown in Table 13.2.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 13.1 – Scheduling Parameter Types

C-Language	Ada-Language
int	`Scheduling_Priority`
sched_param	`Scheduling_Parameters`

Table 13.2 – Scheduling Parameter Operations

C-Language Member	Ada-Language	
	Get Value	Set Value
sched_priority	`Get_Priority`	`Set_Priority`

13.1.1.3 References

sched_param, P1003.4 § 13.1; *sched_priority*, P1003.4 § 13.1.

13.2 Scheduling Policies

13.2.1 Scheduling Policies

13.2.1.1 Synopsis

```
type Scheduling_Policy is range impl.-defined static range;
FIFO_Within_Priority:         constant Scheduling_Policy
  := impl.-defined static const.;
Round_Robin_Within_Priority: constant Scheduling_Policy
  := impl.-defined static const.;
Other_Scheduling:             constant Scheduling_Policy
  := impl.-defined static const.;
```

13.2.1.2 Description

The Ada-language names for constants of the type `Scheduling_Policy` shall correspond to constants of the P1003.4 C-Language Specification as shown in Table 13.3.

Table 13.3 – Scheduling Policies

C-Language	Ada-Language
SCHED_FIFO	`FIFO_Within_Priority`
SCHED_RR	`Round_Robin_Within_Priority`
SCHED_OTHER	`Other_Scheduling`

13.2.1.3 References

SCHED_FIFO, SCHED_RR, SCHED_OTHER, P1003.4 § 13.2.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

13.3 Process Scheduling Functions

13.3.1 Set Scheduling Parameters

13.3.1.1 Synopsis

```
procedure Set_Scheduling_Parameters
   (Process:    in POSIX_Process_Identification.Process_ID;
    Parameters: in Scheduling_Parameters);
procedure Set_Scheduling_Parameters
   (Parameters: in Scheduling_Parameters);
```

13.3.1.2 Description

The Ada-language operations for setting scheduling parameters of a process shall correspond to the P1003.4 C-Language Specification operation as shown in Table 13.4.

Table 13.4 – Set Scheduling Parameters

C-Language			Ada-Language			Notes
sched_setparam			Set_Scheduling_Parameters			
	pid ($\neq 0$)	*(I)*		Process	*(I)*	
	param	*(I)*		Parameters	*(I)*	
sched_setparam			Set_Scheduling_Parameters			
	pid ($= 0$)	*(I)*		*(none)*	*(I)*	(1)
	param	*(I)*		Parameters	*(I)*	

NOTES:

(1) In the case where $pid = 0$, the operation applies to the calling process.

13.3.1.3 References

sched_setparam, P1003.4 § 13.3.1.

13.3.2 Get Scheduling Parameters

13.3.2.1 Synopsis

```
function Get_Scheduling_Parameters
   (Process:    in POSIX_Process_Identification.Process_ID)
  return Scheduling_Parameters;
function Get_Scheduling_Parameters
  return Scheduling_Parameters;
```

13.3.2.2 Description

The Ada-language operations for getting scheduling parameters of a process shall correspond to the P1003.4 C-Language Specification operation as shown in Table 13.5.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 13.5 – Get Scheduling Parameters

C-Language		Ada-Language	
sched_getparam		Get_Scheduling_Parameters	
pid ($\neq 0$)	*(I)*	Process	*(I)*
<*return-value*>		<*return-value*>	
sched_getparam		Get_Scheduling_Parameters	
pid ($= 0$)	*(I)*	*(none)*	*(I)*
<*return-value*>		<*return-value*>	

13.3.2.3 References

sched_getparam, P1003.4 § 13.3.2.

13.3.3 Set Scheduling Policy and Scheduling Parameters

13.3.3.1 Synopsis

```
procedure Set_Scheduling_Policy
   (Process:    in POSIX_Process_Identification.Process_ID;
    Policy:     in Scheduling_Policy;
    Parameters: in Scheduling_Parameters);
procedure Set_Scheduling_Policy
   (Policy:     in Scheduling_Policy;
    Parameters: in Scheduling_Parameters);
```

13.3.3.2 Description

The Ada-language operations for setting the scheduling policy and scheduling parameters of a process shall correspond to the P1003.4 C-Language Specification operation as shown in Table 13.6. D

Table 13.6 – Set Scheduling Policy and Scheduling Parameters

C-Language		Ada-Language	
sched_setscheduler		Set_Scheduling_Policy	
pid ($\neq 0$)	*(I)*	Process	*(I)*
policy	*(I)*	Policy	*(I)*
param	*(I)*	Parameters	*(I)*
sched_setscheduler		Set_Scheduling_Policy	
pid ($= 0$)	*(I)*	*(none)*	*(I)*
policy	*(I)*	Policy	*(I)*
param	*(I)*	Parameters	*(I)*

13.3.3.3 References

sched_setscheduler, P1003.4 § 13.3.3.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

13.3.4 Get Scheduling Policy

13.3.4.1 Synopsis

```
   function Get_Scheduling_Policy
      (Process: POSIX_Process_Identification.Process_ID)
     return Scheduling_Policy;
   function Get_Scheduling_Policy return Scheduling_Policy;
```

13.3.4.2 Description

The Ada-language operations for getting the scheduling policy of a process shall correspond to the P1003.4 C-Language Specification operation as shown in Table 13.7.

Table 13.7 – Get Scheduling Policy

C-Language		Ada-Language	

sched_getscheduler
pid ($\neq 0$)
<*return-value*>
	(I)	`Get_Scheduling_Policy`
		`Process`
		<*return-value*>
			(I)

sched_getscheduler *pid* ($= 0$) <*return-value*>	*(I)*	`Get_Scheduling_Policy` *(none)* <*return-value*>	*(I)*

13.3.4.3 References

sched_getscheduler, P1003.4 § 13.3.4.

13.3.5 Yield CPU

13.3.5.1 Synopsis

```
   procedure Yield;
```

13.3.5.2 Description

The Ada-language operation for yielding the processor to an equal-priority process shall correspond to the P1003.4 C-Language Specification operation as shown in Table 13.8.

Table 13.8 – Yield CPU

C-Language	Ada-Language
sched_yield	`Yield`

13.3.5.3 References

sched_yield, P1003.4 § 13.3.5.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

13.3.6 Get Scheduling Parameter Limits

13.3.6.1 Synopsis

```
function Get_Max_Priority(Policy: Scheduling_Policy)
   return Scheduling_Priority;
function Get_Min_Priority(Policy: Scheduling_Policy)
   return Scheduling_Priority;
function Get_Round_Robin_Interval
   (Process: POSIX_Process_Identification.Process_ID)
   return POSIX.Timespec;
function Get_Round_Robin_Interval
   return POSIX.Timespec;
```

13.3.6.2 Description

The Ada-language operations for gettting scheduling parameter limits shall correspond to the P1003.4 C-Language Specification operations as shown in Table 13.9.

Table 13.9 – Get Scheduling Parameter Limits

C-Language		Ada-Language	
sched_get_priority_max		Get_Max_Priority	
policy	*(I)*	Priority	*(I)*
<return-value>		*<return-value>*	
sched_get_priority_min		Get_Min_Priority	
policy	*(I)*	Priority	*(I)*
<return-value>		*<return-value>*	
sched_rr_get_interval		Get_Round_Robin_Interval	
pid ($\neq 0$)	*(I)*	Process	*(I)*
interval	*(O)*	*<return-value>*	*(O)*
sched_rr_get_interval		Get_Round_Robin_Interval	
pid ($= 0$)	*(I)*	*(none)*	*(I)*
interval	*(O)*	*<return-value>*	*(O)*

13.3.6.3 References

sched_get_priority_max, *sched_get_priority_min*, *sched_get_rr_interval*, P1003.4 §13.3.6.

13.4 Rationale for Execution Scheduling

13.4.1 Packaging

The operations to get the minimum and maximum priority should perhaps be with the other limits, in the package POSIX_Configurable_System_Limits, or perhaps some of the other limits should be scattered with the packages to which they apply.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

13.4.2 Relationship to Ada Priorities

Priorities here are intentionally `Integer`, for consistency with the C interface and for consistency with the precedent established by Ada, for task priorities. Note also that the ranges of priorities supported by the various scheduling policies is not known statically, so defining bounds on this range is not likely to be helpful.

The anticipated practice is for a user to call the functions `Get_Max_Priority` and `Get_Min_Priority` to obtain upper and lower bounds that can be used to define a subtype, which is then used within the application.

13.4.3 P1003.4a Considerations

With P1003.4a, the same priority scheduling models apply to threads. If Ada tasks are implemented as POSIX threads, this will constrain the scheduling semantics of Ada tasks to use the scheduling provided by the P1003.4a implementation. Unfortunately, P1003.4a allows an implementation to support global-only or local-only thread scheduling. Thus, the effects of priority scheduling on tasks will depend on which of these options is supported. There are several other open issues related to P1003.4a thread scheduling, including how priority inheritance fits into the priority scheduling model. Some distinction between base priority and active priority, as is made in the proposed Real-Time Systems annex for Ada 9X, probably will be necessary.

13.4.4 Ada 9X Considerations

With Ada 9X and P1003.4a together, the semantics issues with regard to thread scheduling will change. The Ada 9X mapping proposals call for moving the priority task scheduling model to an annex. Thus, an Ada implementation for P1003.4a may provide its own priority scheduling model without concern for conflicts with the Ada priority model. The Ada 9X Real-Time Annex of [8] specifies a priority scheduling model that is very close to SCHED_FIFO, but which includes so-called priority ceiling emulation (PCE) locking of protected records. Presently, P1003.4a includes (optional) support for PCE on mutexes.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

227 *This side is intentionally blank.*

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Section 14: Clocks and Timers

This section describes the timing facilities available under this standard.

Specification of Package POSIX_Timers

```
with POSIX;                                -- from P1003.5
with POSIX_Signals;                        -- from P1003.5
package POSIX_Timers is
   -- §14.1.2 Type Definitions ...
   -- §14.1.3 Manifest Constants ...
   -- §14.2.1 Clocks ...
   -- §14.2.2 Create a Per-Process Timer ...
   -- §14.2.3 Delete a Per-Process Timer ...
   -- §14.2.4 Per-Process Timers ...
   -- §14.2.5 High Resolution Sleep ...
private
   -- impl.-defined;
end POSIX_Timers;
```

14.1 Data Definitions for Clocks and Timers

14.1.1 Time Value Specification Structures

There is no Ada-language type corresponding to the *itimerspec* type of the P1003.4 C-Language Specification.

14.1.1.1 References

itimerspec, P1003.4 §14.1.1.

14.1.2 Type Definitions

14.1.2.1 Synopsis

```
   type Timer_ID is private;
   type Clock_ID is private;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

14.1.2.2 Description

Clocks are used to measure the passage of time. Timers are based on clocks, and can be used to trigger signals.

The Ada-language names of the types associated with clocks and timers shall correspond to the types of the P1003.4 C-Language Specification as shown in Table 14.1.

Table 14.1 – Timer Types

C-Language	Ada-Language	Notes
timer_t	Timer_ID	
clockid_t	Clock_ID	
timespec	Timespec	(1)

NOTES:

(1) Timespec is defined in the POSIX package.

14.1.3 Manifest Constants

14.1.3.1 Synopsis

```
Clock_Realtime: constant Clock_ID;
```

14.1.3.2 Description

The Ada-language name of the constant for the system wide real-time clock shall correspond to the constant of the P1003.4 C-Language Specification as shown in Table 14.2.

Table 14.2 – Timer Constants

C-Language	Ada-Language
CLOCK_REALTIME	Clock_Realtime

14.2 Clock and Timer Functions

14.2.1 Clocks

14.2.1.1 Synopsis

```
procedure Set_Time
   (Clock: in Clock_ID := Clock_Realtime;
    Value: in POSIX.Timespec);
function Get_Time
   (Clock: Clock_ID := Clock_Realtime)
    return POSIX.Timespec;
function Get_Resolution
   (Clock: Clock_ID := Clock_Realtime)
    return POSIX.Timespec;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

14.2.1.2 Description

Clocks measure the passage of time. A clock may measure real time, or some other implementation-defined concept of time, such as the CPU time used by the process.

The Ada-language operations to read and modify the state of a clock shall correspond to operations of the P1003.4 C-Language Specification as shown in Table 14.3.

Table 14.3 – Clocks

C-Language			Ada-Language	
clock_settime			`Set_Time`	
	clock_id	*(I)*	`Clock`	*(I)*
	tp	*(I)*	`Value`	*(I)*
clock_gettime			`Get_Time`	
	clock_id	*(I)*	`Clock`	*(I)*
	tp	*(O)*	*<return-value>*	*(O)*
clock_getres			`Get_Resolution`	
	clock_id	*(I)*	`Clock`	*(I)*
	res	*(O)*	*<return-value>*	*(O)*

14.2.1.3 Error Handling

If any of these operations fails, `POSIX_Error` shall be raised and the error code shall be as specified in the P1003.4 C-Language Specification, except for the following case:

A `Timespec` value with a *nsec* component outside the range `0 .. 999_999_999` is required to raise `Constraint_Error`.

14.2.1.4 References

clock_settime, *clock_gettime*, *clock_getres*, P1003.4 § 14.2.1.

14.2.2 Create a Per-Process Timer

14.2.2.1 Synopsis

```
function Create_Timer
   (Clock: in Clock_ID;
    Event: in POSIX_Signals.Signal_Event)
  return Timer_ID;
function Create_Timer
   (Clock: in Clock_ID)
  return Timer_ID;
```

14.2.2.2 Description

The Ada-language operations to create a per-process timer shall correspond to operations of the P1003.4 C-Language Specification as shown in Table 14.4.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 14.4 – Per-Process Timers

C-Language			Ada-Language	
timer_create			Create_Timer	
	clock_id	*(I)*	Clock	*(I)*
	evp ($\neq 0$)	*(I)*	Event	*(I)*
	timerid	*(O)*	<*return-value*>	*(O)*
timer_create			Create_Timer	
	clock_id	*(I)*	Clock	*(I)*
	evp (= NULL)	*(I)*	*(none)*	*(I)*
	timerid	*(O)*	<*return-value*>	*(O)*

If the Ada implementation uses any timers, the value of {TIMER_MAX} visible to applications must be adjusted to reflect this.

14.2.2.3 Error Handling

If this operation fails, `POSIX_Error` shall be raised and the error code shall be as specified in the P1003.4 C-Language Specification, except for the following case:

If the application attempts to create a timer that delivers a signal that is reserved for the Ada implementation (see P1003.5 § 3), `POSIX_Error` shall be raised and the error code shall be `Invalid_Argument`. Note that this includes the case where the default signal is specified, and the default signal is `Signal_Alarm`. In particular, calling the version of `Create_Timer` without `Event` parameter and with the value `Clock_Realtime` for the `Clock` parameter shall result in `POSIX_Error` being raised.

14.2.2.4 References

timer_create, P1003.4 § 14.2.2.

14.2.3 Delete a Per-Process Timer

14.2.3.1 Synopsis

```
procedure Delete_Timer(Timer: in Timer_ID);
```

14.2.3.2 Description

The Ada-language operation to delete a per-process timer shall correspond to the operation of the P1003.4 C-Language Specification as shown in Table 14.5.

Table 14.5 – High Resolution Sleep

C-Language			Ada-Language	
timer_delete			Delete_Timer	
	timerid	*(I)*	Timer	*(I)*

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

14.2.3.3 References

timer_delete, P1003.4 § 14.2.3.

14.2.4 Per-Process Timers

14.2.4.1 Synopsis

```
   procedure Get_Times
      (Timer:    in Timer_ID;
       Value:    out POSIX.Timespec;
       Interval: out POSIX.Timespec);
   procedure Set_Absolute
      (Timer:        in  Timer_ID;
       New_Value:    in  POSIX.Timespec;
       New_Interval: in  POSIX.Timespec;
       Old_Value:    out POSIX.Timespec;
       Old_Interval: out POSIX.Timespec);
   procedure Set_Relative
      (Timer:        in  Timer_ID;
       New_Value:    in  POSIX.Timespec;
       New_Interval: in  POSIX.Timespec;
       Old_Value:    out POSIX.Timespec;
       Old_Interval: out POSIX.Timespec);
   function Get_Overruns
      (Timer: Timer_ID) return Natural;
```

14.2.4.2 Description

The Ada-language operations to read and modify the state of a per-process timer shall correspond to the operations of the P1003.4 C-Language Specification as shown in Table 14.6.

Note that if a nanosecond value outside the range `0 .. 999_999_999` is specified, the Ada language implementation is required to raise `Constraint_Error`.

14.2.4.3 References

timer_settime, P1003.4 § 14.2.4; *timer_gettime*, P1003.4 § 14.2.4; *timer_getoverrun*, P1003.4 § 14.2.4.

14.2.5 High Resolution Sleep

14.2.5.1 Synopsis

```
   procedure Nanosleep
      (Request:     in  POSIX.Timespec;
       Residue:     out POSIX.Timespec;
       Interrupted: out Boolean);
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 14.6 – Delete a Per-Process Timer

C-Language		Ada-Language	
timer_gettime		`Get_Times`	
timerid	*(I)*	`Timer`	*(I)*
value.it_value	*(O)*	`Value`	*(O)*
value.it_interval	*(O)*	`Interval`	*(O)*
timer_settime		`Set_Absolute`	
timerid	*(I)*	`Timer`	*(I)*
flags (= TIMER_ABSTIME)	*(I)*	*(none)*	*(I)*
value.it_value	*(I)*	`New_Value`	*(I)*
value.it_interval	*(I)*	`New_Interval`	*(I)*
ovalue.it_value	*(O)*	`Old_Value`	*(O)*
ovalue.it_interval	*(O)*	`Old_Interval`	*(O)*
timer_settime		`Set_Relative`	
timerid	*(I)*	`Timer`	*(I)*
flags (= 0)	*(I)*	*(none)*	*(I)*
value.it_value	*(I)*	`New_Value`	*(I)*
value.it_interval	*(I)*	`New_Interval`	*(I)*
ovalue.it_value	*(O)*	`Old_Value`	*(O)*
ovalue.it_interval	*(O)*	`Old_Interval`	*(O)*
timer_getoverrun		`Set_Overruns`	
timerid	*(I)*	`Timer`	*(I)*
<*return-value*>		<*return-value*>	

14.2.5.2 Description

The Ada-language operation to wait for a specified time shall correspond to the operation of the P1003.4 C-Language Specification as shown in Table 14.7.

Table 14.7 – High Resolution Sleep

C-Language		Ada-Language		Notes
nanosleep		`Nanosleep`		
rqtp	*(I)*	`Request`	*(I)*	
rmtp	*(O)*	`Residue`	*(O)*	
<*return-value*>	*(O)*	`Interrupted`	*(O)*	(1)

NOTES:

(1) The value returned by the parameter `Interrupted` shall be `True` if and only if the operation is interrupted before the requested time has expired.

In an Ada multitasking implementation, this operation shall suspend the execution of at least the calling task until the time expires or the operation is interrupted.

14.2.5.3 References

nanosleep, P1003.4 § 14.2.5.

14.3 Rationale for Timers and Clocks

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Ada provides some standard timing facilities; i.e. the `Duration` type, the `Calendar` package, and the **`delay`** statement. There are known problems with these, including:

1. Absolute and periodic delays are not supported in a reliable form.
2. Periodic time notification is not supported; especially, not in an asynchronous preemptive form.
3. The Ada language requirement for `Duration` to cover a span of at least one day means the granularity attainable with a 32-bit value is unacceptably coarse for some real-time applications.
4. The resolution of delays and the clock are not defined.
5. Only one time reference is provided.
6. There is no provision for setting the clock.

This section provides operations that can help to solve some of these problems.

One central issue for this section is how to reconcile these extended clock and timer features with the standard features of Ada. In particular, it would be nice if we could require that the standard Ada time operations be implemented using the POSIX real-time operations. However, as discussed further below, there are reasons not to do this.

A second central issue is that the P1003.4 timers are *per process*, whereas the Ada delay facility is *per task*. Unless the Ada implementation is on a system that supports POSIX threads (as in P1003.4a), we cannot assume the underlying system will be able to provide these services on a per task basis; thus, in general, we cannot know whether `Nanosleep` will suspend one task or the whole process. Of course, the implementation of the Ada interface may attempt to provide more or less equivalent functionality in some other way. For example, a high resolution timer that generates a signal caught by the Ada RTS could be used to implement a high resolution delay for Ada tasks. The question is whether this can be provided economically, and to which operations it should be applied.

A third issue is that the time delivery mechanism uses the `Signal_Alarm` signal. P1003.5 decided to not allow an application to provide a handler for this signal, since it would almost certainly be in use by the Ada runtime system to implement the standard **`delay`** and `Calendar.Clock` operations. We have chosen to treat any attempt to create a timer that delivers `Signal_Alarm` as an error. A user of timers must specify some other signal.

Lacking really good solutions to these problems, we have chosen to follow the C language binding rather closely.

14.3.1 `Itimerspec`

No Ada type is provided corresponding to *itimerspec*. This is because this type is only used as parameter to a few operations. It seems less cumbersome to simply pass the

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

two component values (the value and interval) as separate parameters.

14.3.2 Read Clock and Set Clock

Note that the `Get_Time` operation is more general than the Ada `Calendar.Clock` function, since it supports multiple clocks.

14.3.3 Set Clock

There have been complaints from many real-time application developers that Ada provides no standard way to reset `Calendar.Clock`. It is recommended that the `Calendar.Clock` function be implemented via `Get_Time` for `Clock_Realtime`. This would then permit applications which read `Calendar.Clock` to reset that clock via `Set_Time`.

14.3.4 High Resolution Sleep

We considered omitting the the high resolution sleep operation, *nanosleep*, since whether this suspends the whole process or just the calling task will be implementation dependent. (This variation will occur even among systems that support POSIX threads, as in P1003.4a.) If the Ada runtime system does preemptive scheduling of tasks, it is likely to do a context switch if this operation is interrupted by a signal. Thus, the thread of control that did the *nanosleep* operation may not regain control until much after the operation actually expires. For similar reasons, the *sleep* operation is not included in the P1003.5 interface.

However, `Nanosleep` can support longer and shorter sleep durations than are normally possible with the standard Ada **`delay`** statement. Therefore, it seems desirable to provide this functionality to the Ada application by some means.

There seem to be three ways of resolving this:

1. Portable programs can be required to not depend on whether one task or all tasks are suspended. This is the solution chosen.

2. The Ada language implementation could be required to ensure that all tasks are suspended during the `Nanosleep`. This would provide portable semantics, but with loss of functionality for those systems where the normal semantics is per-task blocking.

3. Generalize this functionality so that it applies to tasks (rather than just entire processes). If this is done, the facility would probably be implemented using signals, rather than through *nanosleep*.

No exception corresponding to the *errno* value [EINVAL] needs to be raised by the operation, since the Ada compiler will raise `Constraint_Error` if the nanosecond value of the `Time` argument is out of range.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

There is a problem with the parameter *RMTP* (`Residue`). No value is returned in this unless the *nanosleep* operation is interrupted by a signal. For interrupted system calls, P1003.5 normally specifies that `POSIX_Error` is raised with error code `Interrupted_Operation`. However, Ada semantics do not allow scalar **out** parameters to be modified by an operation that propagates an exception. The following alternatives were considered:

1. Use of an access value to point to the location where the out-value is to be returned, as in the C-language interface.
2. Requiring the implementation to use call-by-reference for the *RMTP* parameter, and to update the value before propagating `POSIX_Error`.
3. Modification of the interface to return a Boolean value indicating the operation was interrupted, rather than raising an exception.

The last alternative was chosen, on the grounds that it best preserves the intended functionality of the operation.

Given this choice, there remains the question of whether to include a `Masked_Signals` parameter, to permit masking undesired signals. There are two reasons for providing such a parameter on interruptable operations. The main one is that the ordinary P1003.5 signal interface does not permit a user to handle or mask certain signals, since they are reserved for use by the Ada language implementation. Without any means to mask off such signals, an operation like `Open` may never get a chance to complete. The second reason for providing this parameter is to remind the Ada user, in a very visible way, that the operation is interruptable. In the case of `Nanosleep`, the first reason does not apply; there is no question of whether the operation can "complete" in the sense of operations like `Open`; if it is interrupted, the user can simply repeat it with an adjusted time. The second reason does not apply, either, since the `Interrupted` parameter makes it very clear that the user must take into account the possibility that the operation may be interrupted before the delay is complete.

We considered adding a high resolution sleep until an absolute time, to meet the needs of real time applications. The problem these applications face is task preemption between computing a relative time to sleep and initiating the `Nanosleep`. However, it may be too demanding of an implementation to provide such functionality. One method for implementing an absolute time nanosleep is to use a timer with an absolute time offset in conjunction with sleep. Another method is to use the *pthread_cond_timed_wait* function on an implementation that supports POSIX threads. The decision on whether to include an absolute time nanosleep will rely on whether such functionality is specified in the 1003.4 standard.

14.3.5 Notification of Overruns

It is desirable for a task that has overrun a deadline to be notified and to be able to take recovery action. The C language binding permits this through the *setjmp* and *longjmp* operations used in connection with asynchronous event notification.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

There is a gap in the Ada interface now. An asynchronous event notification scheme is provided, but there is no associated mechanism for transfer of control. With P1003.4a, there is the concept of per-task signal state and sending a signal to a specific task. In this context, it makes sense to allow a signal handler associated with a particular task to propagate an exception to that task.

14.3.6 P1003.4a Considerations

As mentioned above, P1003.4a signals can be sent to individual threads. It will be interesting to see whether P1003.4a also extends the timer facility to per-thread timers.

P1003.4a raises another interesting issue related to timers. The *pthread_cond_timedwait* operation provides a form of time-out. It is not clear how an implementation of threads without special kernel support (such as P1003.4a is supposed to allow) can provide this time-out capability without using a signal to notify the threads implementation of the time-out. Thus, the implementation of *pthread_cond_timedwait* may need to reserve the use of some standard POSIX feature, in much the same way as P1003.5 assumes the use of SIGALRM is reserved for the Ada runtime system.

14.3.7 Ada 9X Considerations

The timer model here is logically similar to a per-task time accounting facility in the Real-Time Annex of [8]. The differences are that the facility in the Ada 9X: (1) measures processor time; (2) measures time per-task, instead of per-process; (3) does not keep track of overruns. It might be worth while to try to unify these two models. In fact, the P1003.4 working group is currently discussing such a proposal, for inclusion as part of P1003.4b.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Section 15: Interprocess Message Passing

This section describes the facilities for interprocess communication (IPC) message passing available under this standard.

Specification of Package `POSIX_Message_Passing`

```
with POSIX;                              -- from P1003.5
with POSIX_File_Status;                  -- from P1003.5
with POSIX_Permissions;                  -- from P1003.5
with POSIX_IO;                           -- from P1003.5
with POSIX_Signals;                      -- from P1003.5
package POSIX_Message_Passing is
   -- §15.1.1 Data Structures   ...
   -- §15.2.1 Open a Message Queue   ...
   -- §15.2.2 Close a Message Queue   ...
   -- §15.2.3 Remove a Message Queue   ...
   -- §15.2.4 Send a Message to a Message Queue   ...
   -- §15.2.5 Receive a Message from a Message Queue   ...
   -- §15.2.6 Notify Process that a Message is Available on a Queue   ...
   -- §15.2.7 Set Message Queue Attributes   ...
   -- §15.2.8 Get Message Queue Attributes   ...
private
   -- impl.-defined;
end POSIX_Message_Passing;
```

15.1 Data Definitions for Message Queues

15.1.1 Data Structures

15.1.1.1 Synopsis

```
   type Message_Queue_Descriptor is private;
   type Message_Queue_Attributes is private;
   type Message_Priority is range impl.-defined static range;
   type Message_Queue_Option_Set is new POSIX.Option_Set;
   Message_Queue_Non_Blocking: constant Message_Queue_Option_Set:= impl.-defined;
   procedure Set_Max_Messages
      (Attrs: in out Message_Queue_Attributes;
       Value: Natural);
   function Get_Max_Messages
      (Attrs: in Message_Queue_Attributes) return Natural;
   procedure Set_Message_Length
      (Attrs: in out Message_Queue_Attributes;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
38          Value: Natural);
39       function Get_Message_Length
40          (Attrs: in Message_Queue_Attributes) return Natural;
41       procedure Set_Options
42          (Attrs: in out Message_Queue_Attributes;
43           Value: Message_Queue_Option_Set);
44       function Get_Options
45          (Attrs: in Message_Queue_Attributes) return Message_Queue_Option_Set;
46       function Get_Message_Count
47          (Attrs: in Message_Queue_Attributes) return Natural;
48       function Get_Sender_Count
49          (Attrs: in Message_Queue_Attributes) return Natural;
50       function Get_Receiver_Count
51          (Attrs: in Message_Queue_Attributes) return Natural;
```

52 15.1.1.2 Description

53 The Ada-language names of the types associated with message queues shall corre-
54 spond to types of the P1003.4 C-Language Specification as shown in Table 15.1.

55 **Table 15.1 – Message Queue Types**

56

C-Language	Ada-Language	Notes
57 *mqd_t*	58 `Message_Queue_Descriptor`	
59 *mq_attr*	60 `Message_Queue_Attributes`	
61 *size_t*	62 `Natural`	
63 *mode_t*	64 `Permission_Set`	65 (1)

66 NOTES:

67 (1) `Permission_Set` is declared in the `POSIX_Permissions` package.

68 The Ada-language names of the constant for specifying that operations on a message
69 queue are non-blocking shall correspond to the constant of the P1003.4 C-Language
70 Specification as shown in Table 15.2.

71 **Table 15.2 – Message Queue Constants**

72

C-Language	Ada-Language
73 MQ_NONBLOCK	`Message_Queue_Non_Blocking`

74 The Ada-language names of the read and write operations for components of objects
75 of type `Message_Queue_Attributes` shall correspond to components of type *mq_attr*
76 in the P1003.4 C-Language Specification as shown in Table 15.3.

77 15.1.1.3 References

78 *mq_attr*, *mqd_t*, P1003.4 § 15.1.1; *mode_t*, P1003.1 § 2.5.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 15.3 – Message Queue Attribute Buffer Operations

C-Language Member	Ada-Language Get Value	Ada-Language Set Value
mq_maxmsg	Get_Max_Messages	Set_Max_Messages
mq_msgsize	Get_Message_Length	Set_Message_Length
mq_flags	Get_Options	Set_Options
mq_curmsgs	Get_Message_Count	*(none)*
mq_sendwait	Get_Sender_Count	*(none)*
mq_rcvwait	Get_Receiver_Count	*(none)*

15.2 Interprocess Message Passing Functions

15.2.1 Open a Message Queue

15.2.1.1 Synopsis

```
function Open
   (Name:    POSIX.POSIX_String;
    Mode:    POSIX_IO.File_Mode;
    Options: POSIX_IO.Open_Option_Set)
  return Message_Queue_Descriptor;
function Open_Or_Create
   (Name:        POSIX.POSIX_String;
    Mode:        POSIX_IO.File_Mode;
    Permissions: POSIX_Permissions.Permission_Set;
    Options:     POSIX_IO.Open_Option_Set)
  return Message_Queue_Descriptor;
function Open_Or_Create
   (Name:        POSIX.POSIX_String;
    Mode:        POSIX_IO.File_Mode;
    Permissions: POSIX_Permissions.Permission_Set;
    Options:     POSIX_IO.Open_Option_Set;
    Attrs:       Message_Queue_Attributes)
  return Message_Queue_Descriptor;
```

15.2.1.2 Description

The Ada-language names of the operations to open or create a message queue shall correspond to the operation of the P1003.4 C-Language Specification as shown in Table 15.4.

15.2.1.3 References

mq_open, P1003.4 § 15.2.1.

15.2.2 Close a Message Queue

15.2.2.1 Synopsis

```
procedure Close(MQ: in Message_Queue_Descriptor);
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 15.4 – Open a Message Queue

C-Language			Ada-Language			Notes
mq_open			`Open`			
	path	*(I)*		`Name`	*(I)*	
	oflag	*(I)*		`Mode`	*(I)*	(1,2)
	oflag	*(I)*		`Options`	*(I)*	(3)
	<return-value>			*<return-value>*		
mq_open			`Open`			
	path	*(I)*		`Name`	*(I)*	
	oflag	*(I)*		`Mode`	*(I)*	(1,5)
	mode	*(I)*		`Permissions`	*(I)*	
	oflag	*(I)*		`Options`	*(I)*	(4)
	attr (= NULL)	*(I)*		*(none)*	*(I)*	
	<return-value>			*<return-value>*		
mq_open			`Open_Or_Create`			
	path	*(I)*		`Name`	*(I)*	
	oflag	*(I)*		`Mode`	*(I)*	(1,5)
	mode	*(I)*		`Permissions`	*(I)*	
	oflag	*(I)*		`Options`	*(I)*	(4)
	attr	*(I)*		`Attrs`	*(I)*	
	<return-value>			*<return-value>*		

NOTES:

(1) Corresponds to components O_RDONLY, O_WRONLY, and O_RDWR.

(2) Components O_CREAT and O_EXCL shall both be clear.

(3) Corresponds to component O_NONBLOCK.

(4) Corresponds to components O_EXCL and O_NONBLOCK.

(5) Component O_CREAT shall be set.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

15.2.2.2 Description

The Ada-language name of the operation to close a message queue shall correspond to the operation of the P1003.4 C-Language Specification as shown in Table 15.5.

Table 15.5 – Close a Message Queue

C-Language	Ada-Language
mq_close *mqdes (I)*	Close MQ (I)

15.2.2.3 References

mq_close, P1003.4 § 15.2.2.

15.2.3 Remove a Message Queue

15.2.3.1 Synopsis

```
procedure Unlink(Name:  in POSIX.POSIX_String);
```

15.2.3.2 Description

The Ada-language name of the operation to destroy a message queue shall correspond to the operation of the P1003.4 C-Language Specification as shown in Table 15.6.

Table 15.6 – Remove a Message Queue

C-Language	Ada-Language
mq_unlink *name (I)*	Unlink Name (I)

15.2.3.3 References

mq_destroy, P1003.4 § 15.2.3.

15.2.4 Send a Message to a Message Queue

15.2.4.1 Synopsis

```
procedure Send
   (MQ:              in Message_Queue_Descriptor;
    Message:         in POSIX.POSIX_String;
    Priority:        in Message_Priority;
    Masked_Signals: in POSIX.Signal_Masking:= POSIX.RTS_Signals);
generic
  type Message_Type is private;
procedure Generic_Send
   (MQ:              in  Message_Queue_Descriptor;
    Message:         in Message_Type;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        Priority:       in Message_Priority;
        Masked_Signals: in  POSIX.Signal_Masking:= POSIX.RTS_Signals);
```

15.2.4.2 Description

The Ada-language name of the operation to send a message to a message queue shall correspond to the operation of the P1003.4 C-Language Specification as shown in Table 15.7.

Table 15.7 – Send a Message to a Message Queue

C-Language		Ada-Language		Notes
mq_send		Send		
mqdes	*(I)*	MQ	*(I)*	
msg_ptr	*(I)*	Message	*(I)*	
msg_len	*(I)*	Message'length	*(I)*	
msg_prio	*(I)*	Priority	*(I)*	
(none)	*(I)*	Masked_Signals	*(I)*	(1)
mq_send		Send		
mqdes	*(I)*	MQ	*(I)*	
msg_ptr	*(I)*	Message	*(I)*	
msg_len	*(I)*	*(none)*	*(I)*	(2)
msg_prio	*(I)*	Priority	*(I)*	
(none)	*(I)*	Masked_Signals	*(I)*	(1)

NOTES:

(1) The Ada parameter `Masked_Signals` shall specify the set of signals that are to be added to the signal mask that is in effect during the operation. The previous signal mask shall be restored when the operation completes.

(2) The generic form must be instantiated with a specific type of message, which determines the length. This type is required to be constrained. Instantiation with an unconstrained array or record type shall raise `Constraint_Error`.

15.2.4.3 References

mq_send, P1003.4 § 15.2.4.

15.2.5 Receive a Message from a Message Queue

15.2.5.1 Synopsis

```
   procedure Receive
      (MQ:             in  Message_Queue_Descriptor;
       Message:        out POSIX.POSIX_String;
       Last:           out Natural;
       Priority:       out Message_Priority;
       Masked_Signals: in  POSIX.Signal_Masking:= POSIX.RTS_Signals);
   generic
     type Message_Type is private;
   procedure Generic_Receive
      (MQ:             in  Message_Queue_Descriptor;
       Message:        out Message_Type;
       Priority:       out Message_Priority;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        Masked_Signals: in  POSIX.Signal_Masking:= POSIX.RTS_Signals);
```

15.2.5.2 Description

The Ada-language name of the operation to receive a message from a message queue shall correspond to the operation of the P1003.4 C-Language Specification as shown in Table 15.8.

Table 15.8 – Receive a Message from a Message Queue

C-Language		Ada-Language		Notes
mq_receive		`Receive`		
mqdes	*(I)*	`MQ`	*(I)*	
msg_ptr	*(O)*	`Message`	*(O)*	
msg_len	*(I)*	`Message'length`	*(I)*	(1)
msg_prio	*(O)*	`Priority`	*(O)*	
(none)	*(I)*	`Masked_Signals`	*(I)*	(3)
<return-value>	*(O)*	`Last`	*(O)*	(2)
mq_receive		`Receive`		
mqdes	*(I)*	`MQ`	*(I)*	
msg_ptr	*(O)*	`Message`	*(O)*	
msg_len	*(I)*	*(none)*	*(I)*	(4)
msg_prio	*(O)*	`Priority`	*(O)*	
(none)	*(I)*	`Masked_Signals`	*(I)*	(4)

NOTES:

(1) `Message'length` provides the length of the user buffer, which is an upper limit on the length of the list that may be returned.

(2) `Last` returns the actual length of the message, which is between zero and `Message'length`.

(3) The Ada parameter `Masked_Signals` shall specify the set of signals that are to be added to the signal mask that is in effect during the operation. The previous signal mask shall be restored when the operation completes.

(4) The generic version must be instantiated for a specific type of message, which determines the anticipated messsage length. An call to an instantiation of the generic operation fails unless the message is of a correct size for the type.

`Generic_Receive` can be instantiated to obtain a procedure for receiving any specified type of message. If the message received is detected to be of a form that cannot be interpreted as a value of the specified type, `Constraint_Error` is raised. In this case, since the message is already removed from the queue when the mismatch is detected, the message is lost.

15.2.5.3 Error Handling

In addition to the error conditions specified in the P1003.4 C-Language Specification, `Constraint_Error` shall be raised by a call to an instantiation of `Generic_Receive` if the message received cannot be interpreted as a value of the specified type.

15.2.5.4 References

mq_receive, P1003.4 § 15.2.5.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

15.2.6 Notify Process that a Message is Available on a Queue

15.2.6.1 Synopsis

```
procedure Request_Notify
   (MQ:    in Message_Queue_Descriptor;
    Event: in POSIX_Signals.Signal_Event);
procedure Remove_Notify(MQ: in Message_Queue_Descriptor);
```

15.2.6.2 Description

The Ada-language names of the operations to request notification of arrival of a message to a message queue and to remove such a request, shall correspond to the operation of the P1003.4 C-Language Specification as shown in Table 15.9.

Table 15.9 – Notify Process that a Message is Available on a Queue

C-Language			Ada-Language			Notes
mq_notify			Request_Notify			
	mqdes	*(I)*		MQ	*(I)*	
	notification ($\neq$ NULL)	*(I)*		Event	*(I)*	
mq_notify			Remove_Notify			
	mqdes	*(I)*		MQ	*(I)*	
	notification (= NULL)	*(I)*		*(none)*	*(I)*	(1)

NOTES:

(1) In the case where *notification* is NULL *mq_notify* has the effect of removing any pending notification request.

15.2.6.3 References

mq_notify, P1003.4 § 15.2.6.

15.2.7 Set Message Queue Attributes

15.2.7.1 Synopsis

```
procedure Set_Attributes
   (MQ:         in  Message_Queue_Descriptor;
    New_Attrs:  in  Message_Queue_Attributes;
    Old_Attrs:  out Message_Queue_Attributes);
```

15.2.7.2 Description

The Ada-language name of the operation to retrieve attributes of a message queue shall correspond to the operation of the P1003.4 C-Language Specification as shown in Table 15.10.

15.2.7.3 References

mq_setattr, P1003.4 § 15.2.7.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 15.10 – Set Message Queue Attributes

C-Language		Ada-Language	
mq_setattr		Set_Attributes	
mqdes	*(I)*	MQ	*(I)*
mqstat	*(I)*	New_Attrs	*(I)*
omqstat	*(O)*	Old_Attrs	*(O)*

15.2.8 Get Message Queue Attributes

15.2.8.1 Synopsis

```
procedure Get_Attributes
   (MQ:     in  Message_Queue_Descriptor;
    Attrs: out Message_Queue_Attributes);
```

15.2.8.2 Description

The Ada-language name of the operation to get attributes of a message queue shall correspond to the operation of the P1003.4 C-Language Specification as shown in Table 15.11.

Table 15.11 – Get Message Queue Attributes

C-Language		Ada-Language	
mq_getattr		Get_Attributes	
mqdes	*(I)*	MQ	*(I)*
mqstat	*(O)*	Attrs	*(O)*

15.2.8.3 References

mq_getattr, P1003.4 § 15.2.8.

15.3 Rationale for Interprocess Message Passing

15.3.1 Message Queue Option Set Type

The `Message_Queue_Option_Set` type has only one value, which is used to specify whether the `Send` and `Receive` operations shall block. However, this type also serves another role; it provides a means for implementations to define additional options for message queues.

15.3.2 Message Queue Opening Not Interruptible

The operation *mq_open* is not interruptible by signals. Therefore the Ada version does not have a `Masked_Signals` parameter. In this respect, it is different from the other "open" operations in POSIX.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

15.3.3 Priorities

Here, it is assumed that the range of message priorities supported by the implementation is not necessarily determined at compile time. The implementation-defined type `Message_Priority` may include more priority values than are supported by the implementation.

15.3.4 Typing of Messages

Since messages generally arrive from other programs, which may be compiled separately and in different languages, reliable type checking of messages is not possible. The generic receive procedure may receive a message of the wrong type, but of a size and format that that is indistinguishable from a message of the anticipated type.

In situations where messages of several types are expected to be received on the same queue, the form of `Receive` with a string buffer should be used. The application will need to parse the message, to infer the type. Unchecked conversion can then be used to convert all or portions of the string into data of the desired Ada type(s).

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Section 16: Realtime Files

This section describes the facilities for managing files to meet real-time requirements available under this standard.

Specification of Package `POSIX_Realtime_Files`

```
with POSIX;                          -- from P1003.5
with POSIX_File_Status;              -- from P1003.5
with POSIX_IO;                       -- from P1003.5
with POSIX_Permissions;              -- from P1003.5
package POSIX_Realtime_Files is
   -- §16.2.1 Realtime Files Specification Structures

   type Component_ID_Set is new POSIX.Option_Set;
      Allocation_Size_Component:    constant Component_ID_Set:= impl.-defined;
      Allocation_Options_Component: constant Component_ID_Set:= impl.-defined;
      Cache_Options_Component:      constant Component_ID_Set:= impl.-defined;
      Direct_IO_Options_Component:  constant Component_ID_Set:= impl.-defined;
      Direct_IO_Transfer_Component: constant Component_ID_Set:= impl.-defined;
      All_Components:               constant Component_ID_Set:= impl.-defined;

   type Increment_Mode is
      (Increment_By_Multiples,
       Increment_By_Exponents,
       Increment_By_List);

   type Allocation_Options is new POSIX.Option_Set;
      Sequential_Allocation:        constant Allocation_Options:= impl.-defined;
      Advise_Sequential_Allocation: constant Allocation_Options:= impl.-defined;

   type Cache_Options is new POSIX.Option_Set;
      Cache_No_Reuse:          constant Cache_Options:= impl.-defined;
      Cache_Random_Access:     constant Cache_Options:= impl.-defined;
      Cache_Sequential_Access: constant Cache_Options:= impl.-defined;

   type Direct_IO_Options is new POSIX.Option_Set;
      Direct_IO_Is_Possible: constant Direct_IO_Options:= impl.-defined;
      Direct_IO_Is_Required: constant Direct_IO_Options:= impl.-defined;

   type Attributes is private;
   function Get_Allocation_Size(Attrbuf: Attributes)
     return POSIX_IO.IO_Offset;
   procedure Set_Allocation_Size
      (Buffer: in out Attributes;
       Value:  in POSIX_IO.IO_Offset);
   function Get_Allocation_Options(Buffer: Attributes)
     return Allocation_Options;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
40      procedure Set_Allocation_Options
41         (Buffer: in out Attributes;
42          Value:  in Allocation_Options);
43      function Get_Cache_Options(Buffer: Attributes)
44        return Cache_Options;
45      procedure Set_Cache_Options
46         (Buffer: in out Attributes;
47          Value:  in Cache_Options);
48      function Get_Direct_IO_Options(Buffer: Attributes)
49        return Direct_IO_Options;
50      procedure Set_Direct_IO_Options
51         (Buffer: in out Attributes;
52          Value:  in Direct_IO_Options);
53      function Get_Direct_IO_Transfer(Buffer: Attributes)
54        return POSIX_IO.IO_Offset;
55      procedure Set_Direct_IO_Transfer
56         (Buffer: in out Attributes;
57          Value:  in POSIX.IO_Count);

58      type Allocation_Capabilities is private;
59      function Get_Min_Size(Buffer: Allocation_Capabilities)
60        return POSIX_IO.IO_Offset;
61      procedure Set_Min_Size
62         (Buffer: in out Allocation_Capabilities;
63          Value:  in POSIX_IO.IO_Offset);
64      function Get_Max_Size(Buffer: Allocation_Capabilities)
65        return POSIX_IO.IO_Offset;
66      procedure Set_Max_Size
67         (Buffer: in out Allocation_Capabilities;
68          Value:  in POSIX_IO.IO_Offset);
69      function Get_Increment(Buffer: Allocation_Capabilities)
70        return POSIX_IO.IO_Offset;
71      procedure Set_Increment
72         (Buffer: in out Allocation_Capabilities;
73          Value:  in POSIX_IO.IO_Offset);
74      function Get_Increment_Mode(Buffer: Allocation_Capabilities)
75        return Increment_Mode;
76      procedure Set_Increment_Mode
77         (Buffer: in out Allocation_Capabilities;
78          Value:  in Increment_Mode);
79      function Get_Options(Buffer: Allocation_Capabilities)
80        return Allocation_Options;
81      procedure Set_Options
82         (Buffer: in out Allocation_Capabilities;
83          Value:  in Allocation_Options);

84      type Cache_Capabilities is private;
85      function Get_Options(Buffer: Cache_Capabilities)
86        return Cache_Options;
87      procedure Set_Options
88         (Buffer: in out Cache_Capabilities;
89          Value:  in Cache_Options);

90      type Buffered_IO_Capabilities is private;
91      function Get_Seek_Alignment(Buffer: Buffered_IO_Capabilities)
92        return POSIX_IO.IO_Offset;
93      procedure Set_Seek_Alignment
94         (Buffer: in out Buffered_IO_Capabilities;
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
95          Value:  in POSIX_IO.IO_Offset);
96      function Get_Min_Transfer(Buffer: Buffered_IO_Capabilities)
97        return POSIX_IO.IO_Offset;
98      procedure Set_Min_Transfer
99         (Buffer: in out Buffered_IO_Capabilities;
100         Value:  in POSIX.IO_Count);
101     function Get_Max_Transfer(Buffer: Buffered_IO_Capabilities)
102       return POSIX.IO_Count;
103     procedure Set_Max_Transfer
104        (Buffer: in out Buffered_IO_Capabilities;
105         Value:  in POSIX.IO_Count);
106     function Get_Increment(Buffer: Buffered_IO_Capabilities)
107       return POSIX.IO_Count;
108     procedure Set_Increment
109        (Buffer: in out Buffered_IO_Capabilities;
110         Value:  in POSIX.IO_Count);
111     function Get_Increment_Mode(Buffer: Buffered_IO_Capabilities)
112       return Increment_Mode;
113     procedure Set_Increment_Mode
114        (Buffer: in out Buffered_IO_Capabilities;
115         Value:  in Increment_Mode);

116     type Async_IO_Capabilities is private;
117     function Get_Min_Transfer(Buffer: Async_IO_Capabilities)
118       return POSIX_IO.IO_Offset;
119     procedure Set_Min_Transfer
120        (Buffer: in out Async_IO_Capabilities;
121         Value:  in POSIX_IO.IO_Offset);
122     function Get_Max_Transfer(Buffer: Async_IO_Capabilities)
123       return POSIX_IO.IO_Offset;
124     procedure Set_Max_Transfer
125        (Buffer: in out Async_IO_Capabilities;
126         Value:  in POSIX_IO.IO_Offset);
127     function Get_Increment(Buffer: Async_IO_Capabilities)
128       return POSIX_IO.IO_Offset;
129     procedure Set_Increment
130        (Buffer: in out Async_IO_Capabilities;
131         Value:  in POSIX_IO.IO_Offset);
132     function Get_Increment_Mode(Buffer: Async_IO_Capabilities)
133       return Increment_Mode;
134     procedure Set_Increment_Mode
135        (Buffer: in out Async_IO_Capabilities;
136         Value:  in Increment_Mode);
137     function Get_Seek_Alignment(Buffer: Async_IO_Capabilities)
138       return POSIX_IO.IO_Offset;
139     procedure Set_Seek_Alignment
140        (Buffer: in out Async_IO_Capabilities;
141         Value:  in POSIX_IO.IO_Offset);
142     function Get_Seek_Boundary(Buffer: Async_IO_Capabilities)
143       return POSIX_IO.IO_Offset;
144     procedure Set_Seek_Boundary
145        (Buffer: in out Async_IO_Capabilities;
146         Value:  in POSIX_IO.IO_Offset);

147     type Direct_IO_Capabilities is private;
148     function Get_Seek_Alignment(Buffer: Direct_IO_Capabilities)
149       return POSIX.IO_Count;
150     procedure Set_Seek_Alignment
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
151        (Buffer: in out Direct_IO_Capabilities;
152         Value:  in POSIX.IO_Count);
153     function Get_Memory_Alignment(Buffer: Direct_IO_Capabilities)
154       return POSIX.IO_Count;
155     procedure Set_Memory_Alignment
156        (Buffer: in out Direct_IO_Capabilities;
157         Value:  in POSIX.IO_Count);
158     function Get_Min_Transfer(Buffer: Direct_IO_Capabilities)
159       return POSIX.IO_Count;
160     procedure Set_Min_Transfer
161        (Buffer: in out Direct_IO_Capabilities;
162         Value:  in POSIX.IO_Count);
163     function Get_Max_Transfer(Buffer: Direct_IO_Capabilities)
164       return POSIX.IO_Count;
165     procedure Set_Max_Transfer
166        (Buffer: in out Direct_IO_Capabilities;
167         Value:  in POSIX.IO_Count);
168     function Get_Increment(Buffer: Direct_IO_Capabilities)
169       return POSIX.IO_Count;
170     procedure Set_Increment
171        (Buffer: in out Direct_IO_Capabilities;
172         Value:  in POSIX.IO_Count);
173     function Get_Increment_Mode(Buffer: Direct_IO_Capabilities)
174       return Increment_Mode;
175     procedure Set_Increment_Mode
176        (Buffer: in out Direct_IO_Capabilities;
177         Value:  in Increment_Mode);
178     function Get_Options(Buffer: Direct_IO_Capabilities)
179       return Direct_IO_Options;
180     procedure Set_Options
181        (Buffer: in out Direct_IO_Capabilities;
182         Value:  in Direct_IO_Options);
183     -- §16.3.1 Create a Realtime File
184     procedure Create_Realtime_File
185        (Name:           in  POSIX.Pathname;
186         Permissions:    in  POSIX_Permissions.Permission_Set;
187         Request:        in  Attributes;
188         Request_Which:  in  Component_ID_Set;
189         Actual:         out Attributes;
190         Actual_Which:   out Component_ID_Set;
191         File:           out POSIX_IO.File_Descriptor;
192         Masked_Signals: in  POSIX.Signal_Masking := POSIX.RTS_Signals);
193     -- §16.3.2 Get Attributes of Realtime File
194     procedure Get_Attributes
195        (File:   in POSIX_IO.File_Descriptor;
196         Actual: out Attributes);
197     -- §16.3.3 Set Attributes of Realtime File
198     procedure Set_Attributes
199        (File:          in  POSIX_IO.File_Descriptor;
200         Request:       in  Attributes;
201         Request_Which: in  Component_ID_Set;
202         Actual:        out Attributes;
203         Actual_Which:  out Component_ID_Set);
204     -- §16.3.4 Get Allocation Capabilities
205     procedure Get_Capabilities
206        (File:   in POSIX_IO.File_Descriptor;
207         Caps:   out Allocation_Capabilities);
208     -- §16.3.5 Get Cache Capabilities
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
209     procedure Get_Capabilities
210        (File:   in POSIX_IO.File_Descriptor;
211         Caps:   out Cache_Capabilities);
212     -- §16.3.6 Get Buffered I/O Capabilities
213     procedure Get_Capabilities
214        (File:   in POSIX_IO.File_Descriptor;
215         Caps:   out Buffered_IO_Capabilities);
216     -- §16.3.7 Get Atomic I/O Capabilities
217     procedure Get_Capabilities
218        (File:   in POSIX_IO.File_Descriptor;
219         Caps:   out Async_IO_Capabilities);
220     -- §16.3.8 Get Direct I/O Capabilities
221     procedure Get_Capabilities
222        (File:   in POSIX_IO.File_Descriptor;
223         Caps:   out Direct_IO_Capabilities);
224     -- §16.3.9 Get Increment Lists

225     type Increment_List_ID is
226        (Allocation_Increment_List,
227         Buffered_IO_Increment_List,
228         Atomic_IO_Increment_List,
229         Direct_IO_Increment_List);

230     type Offset_List is
231       array ( Positive range <> )  of POSIX_IO.IO_Offset;

232     type Count_List is
233       array ( Positive range <> )  of POSIX.IO_Count;
234     procedure Get_Allocation_Increment_List
235        (File:   in POSIX_IO.File_Descriptor;
236         List:   out Offset_List;
237         Last:   out Natural);
238     procedure Get_Increment_List
239        (File:   in POSIX_IO.File_Descriptor;
240         ID:     in Increment_List_ID;
241         List:   out Count_List;
242         Last:   out Natural);

243 private
244     -- impl.-defined;
245 end POSIX_Realtime_Files;
```

246 16.2 Data Definitions for Realtime Files

247 16.2.1 Realtime Files Specification Structures

248 16.2.1.1 Synopsis

```
249     type Component_ID_Set is new POSIX.Option_Set;
250        Allocation_Size_Component:    constant Component_ID_Set:= impl.-defined;
251        Allocation_Options_Component: constant Component_ID_Set:= impl.-defined;
252        Cache_Options_Component:      constant Component_ID_Set:= impl.-defined;
253        Direct_IO_Options_Component:  constant Component_ID_Set:= impl.-defined;
254        Direct_IO_Transfer_Component: constant Component_ID_Set:= impl.-defined;
255        All_Components:               constant Component_ID_Set:= impl.-defined;
256     type Increment_Mode is
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
257        (Increment_By_Multiples,
258         Increment_By_Exponents,
259         Increment_By_List);
260     type Allocation_Options is new POSIX.Option_Set;
261        Sequential_Allocation:        constant Allocation_Options:= impl.-defined;
262        Advise_Sequential_Allocation: constant Allocation_Options:= impl.-defined;
263     type Cache_Options is new POSIX.Option_Set;
264        Cache_No_Reuse:          constant Cache_Options:= impl.-defined;
265        Cache_Random_Access:     constant Cache_Options:= impl.-defined;
266        Cache_Sequential_Access: constant Cache_Options:= impl.-defined;
267     type Direct_IO_Options is new POSIX.Option_Set;
268        Direct_IO_Is_Possible: constant Direct_IO_Options:= impl.-defined;
269        Direct_IO_Is_Required: constant Direct_IO_Options:= impl.-defined;
270     type Attributes is private;
271     function Get_Allocation_Size(Attrbuf: Attributes)
272       return POSIX_IO.IO_Offset;
273     procedure Set_Allocation_Size
274        (Buffer: in out Attributes;
275         Value:  in POSIX_IO.IO_Offset);
276     function Get_Allocation_Options(Buffer: Attributes)
277       return Allocation_Options;
278     procedure Set_Allocation_Options
279        (Buffer: in out Attributes;
280         Value:  in Allocation_Options);
281     function Get_Cache_Options(Buffer: Attributes)
282       return Cache_Options;
283     procedure Set_Cache_Options
284        (Buffer: in out Attributes;
285         Value:  in Cache_Options);
286     function Get_Direct_IO_Options(Buffer: Attributes)
287       return Direct_IO_Options;
288     procedure Set_Direct_IO_Options
289        (Buffer: in out Attributes;
290         Value:  in Direct_IO_Options);
291     function Get_Direct_IO_Transfer(Buffer: Attributes)
292       return POSIX_IO.IO_Offset;
293     procedure Set_Direct_IO_Transfer
294        (Buffer: in out Attributes;
295         Value:  in POSIX.IO_Count);
296     type Allocation_Capabilities is private;
297     function Get_Min_Size(Buffer: Allocation_Capabilities)
298       return POSIX_IO.IO_Offset;
299     procedure Set_Min_Size
300        (Buffer: in out Allocation_Capabilities;
301         Value:  in POSIX_IO.IO_Offset);
302     function Get_Max_Size(Buffer: Allocation_Capabilities)
303       return POSIX_IO.IO_Offset;
304     procedure Set_Max_Size
305        (Buffer: in out Allocation_Capabilities;
306         Value:  in POSIX_IO.IO_Offset);
307     function Get_Increment(Buffer: Allocation_Capabilities)
308       return POSIX_IO.IO_Offset;
309     procedure Set_Increment
310        (Buffer: in out Allocation_Capabilities;
311         Value:  in POSIX_IO.IO_Offset);
312     function Get_Increment_Mode(Buffer: Allocation_Capabilities)
313       return Increment_Mode;
314     procedure Set_Increment_Mode
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
315         (Buffer: in out Allocation_Capabilities;
316          Value:  in Increment_Mode);
317       function Get_Options(Buffer: Allocation_Capabilities)
318         return Allocation_Options;
319       procedure Set_Options
320         (Buffer: in out Allocation_Capabilities;
321          Value:  in Allocation_Options);
322       type Cache_Capabilities is private;
323       function Get_Options(Buffer: Cache_Capabilities)
324         return Cache_Options;
325       procedure Set_Options
326         (Buffer: in out Cache_Capabilities;
327          Value:  in Cache_Options);
328       type Buffered_IO_Capabilities is private;
329       function Get_Seek_Alignment(Buffer: Buffered_IO_Capabilities)
330         return POSIX_IO.IO_Offset;
331       procedure Set_Seek_Alignment
332         (Buffer: in out Buffered_IO_Capabilities;
333          Value:  in POSIX_IO.IO_Offset);
334       function Get_Min_Transfer(Buffer: Buffered_IO_Capabilities)
335         return POSIX_IO.IO_Offset;
336       procedure Set_Min_Transfer
337         (Buffer: in out Buffered_IO_Capabilities;
338          Value:  in POSIX.IO_Count);
339       function Get_Max_Transfer(Buffer: Buffered_IO_Capabilities)
340          return POSIX.IO_Count;
341       procedure Set_Max_Transfer
342         (Buffer: in out Buffered_IO_Capabilities;
343          Value:  in POSIX.IO_Count);
344       function Get_Increment(Buffer: Buffered_IO_Capabilities)
345         return POSIX.IO_Count;
346       procedure Set_Increment
347         (Buffer: in out Buffered_IO_Capabilities;
348          Value:  in POSIX.IO_Count);
349       function Get_Increment_Mode(Buffer: Buffered_IO_Capabilities)
350         return Increment_Mode;
351       procedure Set_Increment_Mode
352         (Buffer: in out Buffered_IO_Capabilities;
353          Value:  in Increment_Mode);
354       type Async_IO_Capabilities is private;
355       function Get_Min_Transfer(Buffer: Async_IO_Capabilities)
356         return POSIX_IO.IO_Offset;
357       procedure Set_Min_Transfer
358         (Buffer: in out Async_IO_Capabilities;
359          Value:  in POSIX_IO.IO_Offset);
360       function Get_Max_Transfer(Buffer: Async_IO_Capabilities)
361         return POSIX_IO.IO_Offset;
362       procedure Set_Max_Transfer
363         (Buffer: in out Async_IO_Capabilities;
364          Value:  in POSIX_IO.IO_Offset);
365       function Get_Increment(Buffer: Async_IO_Capabilities)
366         return POSIX_IO.IO_Offset;
367       procedure Set_Increment
368         (Buffer: in out Async_IO_Capabilities;
369          Value:  in POSIX_IO.IO_Offset);
370       function Get_Increment_Mode(Buffer: Async_IO_Capabilities)
371         return Increment_Mode;
372       procedure Set_Increment_Mode
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
373        (Buffer: in out Async_IO_Capabilities;
374         Value:  in Increment_Mode);
375     function Get_Seek_Alignment(Buffer: Async_IO_Capabilities)
376       return POSIX_IO.IO_Offset;
377     procedure Set_Seek_Alignment
378        (Buffer: in out Async_IO_Capabilities;
379         Value:  in POSIX_IO.IO_Offset);
380     function Get_Seek_Boundary(Buffer: Async_IO_Capabilities)
381       return POSIX_IO.IO_Offset;
382     procedure Set_Seek_Boundary
383        (Buffer: in out Async_IO_Capabilities;
384         Value:  in POSIX_IO.IO_Offset);
385     type Direct_IO_Capabilities is private;
386     function Get_Seek_Alignment(Buffer: Direct_IO_Capabilities)
387       return POSIX.IO_Count;
388     procedure Set_Seek_Alignment
389        (Buffer: in out Direct_IO_Capabilities;
390         Value:  in POSIX.IO_Count);
391     function Get_Memory_Alignment(Buffer: Direct_IO_Capabilities)
392       return POSIX.IO_Count;
393     procedure Set_Memory_Alignment
394        (Buffer: in out Direct_IO_Capabilities;
395         Value:  in POSIX.IO_Count);
396     function Get_Min_Transfer(Buffer: Direct_IO_Capabilities)
397       return POSIX.IO_Count;
398     procedure Set_Min_Transfer
399        (Buffer: in out Direct_IO_Capabilities;
400         Value:  in POSIX.IO_Count);
401     function Get_Max_Transfer(Buffer: Direct_IO_Capabilities)
402       return POSIX.IO_Count;
403     procedure Set_Max_Transfer
404        (Buffer: in out Direct_IO_Capabilities;
405         Value:  in POSIX.IO_Count);
406     function Get_Increment(Buffer: Direct_IO_Capabilities)
407       return POSIX.IO_Count;
408     procedure Set_Increment
409        (Buffer: in out Direct_IO_Capabilities;
410         Value:  in POSIX.IO_Count);
411     function Get_Increment_Mode(Buffer: Direct_IO_Capabilities)
412       return Increment_Mode;
413     procedure Set_Increment_Mode
414        (Buffer: in out Direct_IO_Capabilities;
415         Value:  in Increment_Mode);
416     function Get_Options(Buffer: Direct_IO_Capabilities)
417       return Direct_IO_Options;
418     procedure Set_Options
419        (Buffer: in out Direct_IO_Capabilities;
420         Value:  in Direct_IO_Options);
```

The Ada-language names for data types associated with realtime file attribute buffers shall correspond to data types of the P1003.4 C-Language Specification as shown in Table 16.1.

The Ada-language names for constants of the `Component_ID_Set` type shall correspond to constants of the P1003.4 C-Language Specification as shown in Table 16.2.

The Ada-language names for constants of the `Allocation_Options` type shall corre-

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 16.1 – Realtime Files Specification Structures

C-Language	Ada-Language	Notes
rf_attrbuf	`Attributes`	
int	`Allocation_Options`	
int	`Cache_Options`	
int	`Direct_IO_Options`	
int	`Component_ID_Set`	
off_t	`IO_Offset`	(1)
unsigned	`IO_Offset`	(1)

NOTES:

(1) `IO_Offset` is defined in `POSIX_IO`.

Table 16.2 – Attributes Component Identification Sets

C-Language	Ada-Language
ATB_ALLOC	`Allocation_Size_Component`
ATB_ALLOCFLAGS	`Allocation_Options_Component`
ATB_CACHEFLAGS	`Cache_Options_Component`
ATB_DIOFLAGS	`Direct_IO_Options_Component`
ATB_DIOSIZE	`Direct_IO_Transfer_Component`
ATB_ALL	`All_Components`

spond to constants of the P1003.4 C-Language Specification as shown in Table 16.3.

Table 16.3 – Allocation Options

C-Language	Ada-Language
ATB_SEQ	`Sequential_Allocation`
ATB_ADVSEQ	`Advise_Sequential_Allocation`
ATC_SEQ	`Sequential_Allocation`

The Ada-language names for constants of the `Cache_Options` type shall correspond to constants of the P1003.4 C-Language Specification as shown in Table 16.4.

The Ada-language names for constants of the `Direct_IO_Options` type shall correspond to constants of the P1003.4 C-Language Specification as shown in Table 16.5.

The Ada-language names for read and write operations of the `Attributes` type shall correspond to components of the P1003.4 C-Language Specification type *rf_attrbuf* as shown in Table 16.6.

The Ada-language names for data types associated with realtime file capability buffers shall correspond to data types of the P1003.4 C-Language Specification as shown in Table 16.7.

The Ada-language names for constants of the `Increment_Mode` type shall correspond to constants of the P1003.4 C-Language Specification as shown in Table 16.8.

The Ada-language names for read and write operations of the `Allocation_Capabilities` type shall correspond to components of the P1003.4 C-Language Specification *rf_capallocbuf* type as shown in Table 16.9.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 16.4 – Cache Options

C-Language	Ada-Language
ATB_CACHENOREUSE	Cache_No_Reuse
ATB_CACHERANDOM	Cache_Random_Access
ATB_CACHESEQUENTIAL	Cache_Sequential_Access
ATC_CACHENOREUSE	Cache_No_Reuse
ATC_CACHERANDOM	Cache_Random_Access
ATC_CACHESEQUENTIAL	Cache_Sequential_Access

Table 16.5 – Direct IO Options

C-Language	Ada-Language
ATC_DIOCAPABLE	Direct_IO_Is_Possible
ATC_DIOREQUIRED	Direct_IO_Is_Required
ATB_DIOENABLE	Direct_IO_Is_Required

Table 16.6 – Attributes Component Operations

C-Language Member	Ada-Language	
	Get Value	Set Value
atb_alloc	Get_Allocation_Size	Set_Allocation_Size
atb_allocflags	Get_Allocation_Options	Set_Allocation_Options
atb_cacheflags	Get_Cache_Options	Set_Cache_Options
atb_dioflags	Get_Direct_IO_Options	Set_Direct_IO_Options
atb_diosize	Get_Direct_IO_Transfer	Set_Direct_IO_Transfer

Table 16.7 – Get Cache Capabilities

C-Language	Ada-Language	Notes
rf_capallocbuf	Allocation_Capabilities	
rf_capcachebuf	Cache_Capabilities	
rf_capaiobuf	Atomic_IO_Capabilities	
rf_capdiobuf	Direct_IO_Capabilities	
off_t	IO_Offset	(1)
size_t	Count_List	
int	Increment_Mode	
int	Allocation_Options	
int	Cache_Options	
int	Direct_IO_Options	

NOTES:

(1) `IO_Offset` is defined in the package `POSIX`.

Table 16.8 – Increment Modes

C-Language	Ada-Language
ATC_MULTIPLY	Increment_By_Multiples
ATC_EXPONENT	Increment_By_Exponents
ATC_LIST	Increment_By_List

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 16.9 – Allocation Capabilities Component Operations

C-Language Member	Ada-Language	
	Get Value	Set Value
atc_allocmin	Get_Min_Size	Set_Min_Size
atc_allocmax	Get_Max_Size	Set_Max_Size
atc_allocincr	Get_Increment	Set_Increment
atc_allocincrtype	Get_Increment_Mode	Set_Increment_Mode
atc_allocflags	Get_Options	Set_Options

The Ada-language names for read and write operations of the Cache_Capabilities type shall correspond to components of the P1003.4 C-Language Specification *rf_capcachcebuf* type as shown in Table 16.10.

Table 16.10 – Cache Capabilities Component Operations

C-Language Member	Ada-Language	
	Get Value	Set Value
atc_cacheflags	Get_Cache_Options	Set_Cache_Options

The Ada-language names for read and write operations of the Buffered_IO_Capabilities type shall correspond to components of the P1003.4 C-Language Specification *rf_capbiobuf* type as shown in Table 16.11.

Table 16.11 – Buffered I/O Capabilities Component Operations

C-Language Member	Ada-Language	
	Get Value	Set Value
atc_boffset	Get_Seek_Alignment	Set_Seek_Alignment
atc_biomin	Get_Min_Transfer	Set_Min_Transfer
atc_biomax	Get_Max_Transfer	Set_Max_Transfer
atc_bioincr	Get_Increment	Set_Increment
atc_bioincrtype	Get_Increment_Mode	Set_Increment_Mode

The Ada-language names for read and write operations of the Async_IO_Capabilities type shall correspond to components of the P1003.4 C-Language Specification *rf_capaiobuf* type as shown in Table 16.12.

The Ada-language names for read and write operations of the Direct_IO_Capabilities type shall correspond to components of the P1003.4 C-Language Specification *rf_capdiobuf* type as shown in Table 16.13.

16.2.1.2 References

ATB_ALLOC, ATB_ALLOCFLAGS, ATB_CACHEFLAGS, ATB_DIOFLAGS, ATB_DIOSIZE, ATB_ALL, P1003.4 § 16.2.1; *rf_attrbuf*, P1003.4 § 16.2.1; *rf_capallocbuf*, *rf_capcachebuf*, *rf_capaiobuf*, *rf_capdiobuf*, P1003.4 § 16.2.1.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 16.12 – Atomic I/O Capabilities Component Operations

C-Language Member	Ada-Language	
	Get Value	Set Value
atc_aiomin	Get_Min_Transfer	Set_Min_Transfer
atc_aiomax	Get_Max_Transfer	Set_Max_Transfer
atc_aioincr	Get_Increment	Set_Increment
atc_aioincrtype	Get_Increment_Mode	Set_Increment_Mode
atc_aiosoffset	Get_Seek_Alignment	Set_Seek_Alignment
atc_aioboundary	Get_Seek_Boundary	Set_Seek_Boundary

Table 16.13 – Direct I/O Capabilities Component Operations

C-Language Member	Ada-Language	
	Get Value	Set Value
atc_doffset	Get_Seek_Alignment	Set_Seek_Alignment
atc_dalign	Get_Memory_Alignment	Set_Memory_Alignment
atc_diomin	Get_Min_Transfer	Set_Min_Transfer
atc_diomax	Get_Max_Transfer	Set_Max_Transfer
atc_dioincr	Get_Increment	Set_Increment
atc_dioincrtype	Get_Increment_Mode	Set_Increment_Mode
atc_dioflags	Get_Options	Set_Options

16.3 Realtime File Functions

16.3.1 Create a Realtime File

16.3.1.1 Synopsis

```
procedure Create_Realtime_File
   (Name:           in  POSIX.Pathname;
    Permissions:    in  POSIX_Permissions.Permission_Set;
    Request:        in  Attributes;
    Request_Which:  in  Component_ID_Set;
    Actual:         out Attributes;
    Actual_Which:   out Component_ID_Set;
    File:           out POSIX_IO.File_Descriptor;
    Masked_Signals: in  POSIX.Signal_Masking := POSIX.RTS_Signals);
```

16.3.1.2 Description

The Ada-language operation for creating a realtime file shall correspond to the P1003.4 C-Language Specification operation as shown in Table 16.14.

The `Request_Which` parameter specifies which Attributes are requested, and the `Actual_Which` parameter specifies which attributes are actually granted. Each of these is a set, which can be produced by combining various values of the `Component_ID_Set` type, such as `Allocation_Options + Cache_Options`.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 16.14 – Create a Realtime File

C-Language		Ada-Language		Notes
rf_create		Create_Realtime_File		
path	*(I)*	Name	*(I)*	
mode	*(I)*	Permissions	*(I)*	
reqattr	*(I)*	Request	*(I)*	
reqwhich	*(I)*	Request_Which	*(I)*	
actattr	*(O)*	Actual	*(O)*	
actwhich	*(O)*	Actual_Which	*(O)*	
<return-value>	*(O)*	File	*(O)*	
(none)	*(I)*	Masked_Signals	*(I)*	(1)

NOTES:

(1) The Ada parameter Masked_Signals shall specify the set of signals that are to be added to the signal mask that is in effect during the operation. The previous signal mask shall be restored when the operation completes.

16.3.2 Get Attributes of Realtime File

16.3.2.1 Synopsis

```
procedure Get_Attributes
   (File:   in POSIX_IO.File_Descriptor;
    Actual: out Attributes);
```

16.3.2.2 Description

The Ada-language operation for getting the attributes of a realtime file shall correspond to the P1003.4 C-Language Specification operation as shown in Table 16.15.

Table 16.15 – Get Attributes of Realtime File

C-Language		Ada-Language	
rf_getattr		Get_Attributes	
fildes	*(I)*	File	*(I)*
actattr	*(O)*	Actual	*(O)*

16.3.2.3 References

rf_getattr, P1003.4 § 16.3.2.

16.3.3 Set Attributes of Realtime File

16.3.3.1 Synopsis

```
procedure Set_Attributes
   (File:          in  POSIX_IO.File_Descriptor;
    Request:       in  Attributes;
    Request_Which: in  Component_ID_Set;
    Actual:        out Attributes;
    Actual_Which:  out Component_ID_Set);
```

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

16.3.3.2 Description

The Ada-language operation for setting the attributes of a realtime file shall correspond to the P1003.4 C-Language Specification operation as shown in Table 16.16.

Table 16.16 – Set Attributes of Realtime File

C-Language		Ada-Language	
rf_setattr		`Set_Attributes`	
fildes	*(I)*	`File`	*(I)*
reqattr	*(I)*	`Request`	*(I)*
reqwhich	*(I)*	`Request_Which`	*(I)*
actattr	*(O)*	`Actual`	*(O)*
actwhich	*(O)*	`Actual_Which`	*(O)*

The `Request_Which` parameter specifies which Attributes are requested to be set, and the `Actual_Which` parameter specifies which attributes are actually set.

16.3.3.3 References

rf_setattr, P1003.4 § 16.3.3.

16.3.4 Get Allocation Capabilities

16.3.4.1 Synopsis

```
procedure Get_Capabilities
   (File:   in POSIX_IO.File_Descriptor;
    Caps:   out Allocation_Capabilities);
```

16.3.4.2 Description

The Ada-language operation for getting the allocation capabilities of realtime files shall correspond to the P1003.4 C-Language Specification operation as shown in Table 16.17.

Table 16.17 – Get Allocation Capabilities

C-Language		Ada-Language	
rf_getalloccap		`Get_Capabilities`	
fildes	*(I)*	`File`	*(I)*
capbufp	*(O)*	`Buffer`	*(O)*

16.3.4.3 References

rf_getalloccap, P1003.4 § 16.3.4.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

16.3.5 Get Cache Capabilities

16.3.5.1 Synopsis

```
procedure Get_Capabilities
   (File:   in POSIX_IO.File_Descriptor;
    Caps:   out Cache_Capabilities);
```

16.3.5.2 Description

The Ada-language operation for getting the cache capabilities of realtime files shall correspond to the P1003.4 C-Language Specification operation as shown in Table 16.18.

Table 16.18 – Get Cache Capabilities

C-Language		Ada-Language	
rf_getcachecap		Get_Capabilities	
fildes	*(I)*	File	*(I)*
capbufp	*(O)*	Buffer	*(O)*

16.3.5.3 References

rf_getcachecap, P1003.4 § 16.3.5.

16.3.6 Get Buffered I/O Capabilities

16.3.6.1 Synopsis

```
procedure Get_Capabilities
   (File:   in POSIX_IO.File_Descriptor;
    Caps:   out Buffered_IO_Capabilities);
```

16.3.6.2 Description

The Ada-language operation for getting the buffered I/O capabilities of realtime files shall correspond to the P1003.4 C-Language Specification operation as shown in Table 16.19.

Table 16.19 – Get Buffered I/O Capabilities

C-Language		Ada-Language	
rf_getbiocap		Get_Capabilities	
fildes	*(I)*	File	*(I)*
capbufp	*(O)*	Buffer	*(O)*

16.3.6.3 References

rf_getbiocap, P1003.4 § 16.3.6.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

16.3.7 Get Atomic I/O Capabilities

16.3.7.1 Synopsis

```
procedure Get_Capabilities
   (File:   in POSIX_IO.File_Descriptor;
    Caps:   out Async_IO_Capabilities);
```

16.3.7.2 Description

The Ada-language operation for getting the atomic I/O capabilities of realtime files shall correspond to the P1003.4 C-Language Specification operation as shown in Table 16.20.

Table 16.20 – Get Atomic I/O Capabilities

C-Language		Ada-Language	
rf_getaiocap		Get_Capabilities	
fildes	*(I)*	File	*(I)*
capbufp	*(O)*	Buffer	*(O)*

16.3.7.3 References

rf_getaiocap, P1003.4 § 16.3.7.

16.3.8 Get Direct I/O Capabilities

16.3.8.1 Synopsis

```
procedure Get_Capabilities
   (File:   in POSIX_IO.File_Descriptor;
    Caps:   out Direct_IO_Capabilities);
```

16.3.8.2 Description

The Ada-language operation for getting the direct I/O capabilities of realtime files shall correspond to the P1003.4 C-Language Specification operation as shown in Table 16.21.

Table 16.21 – Get Direct I/O Capabilities

C-Language		Ada-Language	
rf_getdiocap		Get_Capabilities	
fildes	*(I)*	File	*(I)*
capbufp	*(O)*	Buffer	*(O)*

16.3.8.3 References

rf_getdiocap, P1003.4 § 16.3.8.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

16.3.9 Get Increment Lists

16.3.9.1 Synopsis

```
type Increment_List_ID is
   (Allocation_Increment_List,
    Buffered_IO_Increment_List,
    Atomic_IO_Increment_List,
    Direct_IO_Increment_List);
type Offset_List is
  array ( Positive range <> )  of POSIX_IO.IO_Offset;
type Count_List is
  array ( Positive range <> )  of POSIX.IO_Count;
procedure Get_Allocation_Increment_List
   (File:   in POSIX_IO.File_Descriptor;
    List:   out Offset_List;
    Last:   out Natural);
procedure Get_Increment_List
   (File:   in POSIX_IO.File_Descriptor;
    ID:     in Increment_List_ID;
    List:   out Count_List;
    Last:   out Natural);
```

16.3.9.2 Description

The Ada-language names for constants of the `Increment_List_ID` type shall correspond to constants of the P1003.4 C-Language Specification as shown in Table 16.22.

Table 16.22 – Increment List IDs

C-Language	Ada-Language
ATC_ALLOCINCR	`Allocation_Increment_List`
ATC_BIOINCR	`Buffered_IO_Increment_List`
ATC_ATINCR	`Atomic_IO_Increment_List`
ATC_DIOINCR	`Direct_IO_Increment_List`

The Ada-language operation for getting the increment list of a capability attribute of a file shall correspond to the P1003.4 C-Language Specification operation as shown in Table 16.23.

16.3.9.3 References

rf_getincr, P1003.4 § 16.3.9.

16.4 Rationale for Realtime Files

The correspondence of types and operations to the P1003.4 C-Language Specification is very direct. The *rf_attrbuf* and rf_cap...buf types have a very large number of components. Because of the global decision to use private types, with functions and subprograms to read and write components, in places where the P1003.4 C-Language Specification uses record types, there are a great many function and subprogram declarations.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 16.23 – Get Increment Lists

C-Language		Ada-Language		Notes
rf_getallocincr		`Get_Allocation_Increment_List`		
fildes	*(I)*	`File`	*(I)*	
offbuffer	*(O)*	`List`	*(O)*	
buflen	*(I)*	`List'length`	*(I)*	(1)
]	*(O)*	*<return-value>*	*(O)*	Last(2)
rf_getincr		`Get_Increment_List`		
fildes	*(I)*	`File`	*(I)*	
incr	*(I)*	`ID`	*(I)*	
sizebuffer	*(O)*	`List`	*(O)*	
buflen	*(I)*	`List'length`	*(I)*	(1)
<return-value>	*(O)*	`Last`	*(O)*	(2)

NOTES:

(1) `List'length` provides the length of the user buffer, which is an upper limit on the length of the list that may be returned.

(2) `Last` returns the actual length of the list, which is between zero and `List'length`.

Because of the global decision to use names of a style consistent with P1003.5, there are a great many long identifiers. Nevertheless, some abbreviations are used to reduce the length of these identifiers. For example, `Get_Buffered_IO_Seek_Boundary` is used instead of `Get_Buffered_IO_Transfer_Seek_Alignment_Boundary`.

16.4.1 Buffer Allocation

There does not appear to be any need for Ada operations corresponding to the *rf_getbuf* and *rf_freebuf* operations, since Ada does have alignment clauses for record types, **new**, and `Unchecked_Deallocation`. These operations should not be included in the language-independent specification.

16.4.2 Constant Names

In order to control the proliferation of names, in this Ada binding some of the names of the C-language interface have been combined. For example, ATB_ALLOC and ATC_ALLOC have been combined into one constant, Allocation_Options. While these constants may be different in the C-language interface, there is sufficient context for them to combined in the Ada-language interface. If the Ada binding is implemented over a C-language interface, this may in some cases mean values needs to be translated by the Ada implementation.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Section A
(informative)
Bibliography

The following documents are related to this standard in an informative manner, or were used as references in the preparation of this standard.

{1} *Portable Operating System Interface (POSIX) – Part 1: System Application Program Interface (API)* [*C Language*], IEEE (1990).

{2} *Draft Standard for Information Technology – POSIX C Language Bindings – Part 1: Binding for ISO/ IED 9945-1, Draft 2*, IEEE (November 1991).

{3} *Information Technology – Portable Operating System Interface (POSIX) – Part 1: System Application Program Interface (API)* [*Language Independent*], *Draft 2*, IEEE (November, 1991).

{4} *POSIX Ada Language Interface Part I, Binding for System API, Draft 9*, IEEE (May, 1992).

{5} *Realtime Extension for Portable Operating Systems, Draft 13*, IEEE (September, 1992).

{6} *Threads Extension for Portable Operating Systems, Draft 6*, IEEE (February, 1992).

{7} *Military Standard Ada Programming Language*, ANSI/MILSTD1815A, U.S. Department of Defense, Ada Joint Program Office (January 1983).

{8} *Ada 9X Mapping Document, Volume II – Mapping Specification*, Ada 9X Project Report, Office of the Secretary of Defense for Acquisition (December, 1991).

{9} *Realtime Extension for Portable Operating Systems (Language-Independent Specification), Draft 2*, IEEE (February, 1990).

{10} *Ada Uniformity Issues*, the ISO/IEC JTC1 SC22 WG9 Uniformity Rapporteur Group, 1992.

{11} *Ada Issue 00325*, the ISO/IEC JTC1 SCSS WG Ada Rapporteur Group.

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

This side is intentionally blank.

Alphabetic Topical Index

A

address
attribute ... 6, 27, 83, 139
Address
type ... 6, 27, 36–37, 84–85, 139
Address_Error
exception ... 28, 139
Address_Offset
type ... 27, 139
Advise_Sequential_Allocation
constant ... 127, 139
AIO_ALLDONE
constant ... 61, 139
aio_cancel
function ... 61, 139
AIO_CANCELED
constant ... 61, 139
aiocb
type ... 56–57, 139
aio_error
function ... 30, 59–60, 64, 139
aio_fsync
function ... 63–64, 139
{AIO_LISTIO_MAX}
limit ... 20, 139
{AIO_MAX}
limit ... 20, 139
AIO_NOTCANCELED
constant ... 61, 139
{AIO_PRIO_DELTA_MAX}
limit ... 20, 46, 139
aio_read
function ... 57, 139
aio_return
function ... 60, 64, 139
aio_suspend
function ... 62–63, 65, 139
aio_write
function ... 58, 139
Align_To_Page
procedure ... 77–78, 139
All_Components
constant ... 127, 139
Allocation_Capabilities
type ... 127–128, 139
Allocation_Increment_List
enumeration literal ... 135, 139
Allocation_Options
type ... 126–128, 139
Allocation_Options
constant ... 136
Allocation_Options
constant ... 139
Allocation_Options_Component
constant ... 127, 139
Allocation_Size_Component
constant ... 127, 139
Allow_Execute
constant ... 22, 78, 139
Allow_Nothing
constant ... 78, 139
Allow_Read
constant ... 22, 78, 139
Allow_Write
constant ... 22, 78, 87–89, 139
Argument_List_Too_Long
constant ... 15, 139
Async_IO_Blocking_Behavior
subtype ... 21, 139
Async_IO_Capabilities
type ... 129, 139
Async_IO_Is_Supported
function ... 47, 52, 139
Async_IO_Max
function ... 47, 139
Async_IO_Maxima
subtype ... 20, 139
Async_IO_Priority_Delta_Max
function ... 47, 139
Async_IO_Priority_Delta_Maxima
subtype ... 20, 139
Async_IO_Support
subtype ... 18, 139
ATB_ADVSEQ
constant ... 127, 139
ATB_ALL
constant ... 127, 129, 139
ATB_ALLOC
constant ... 127, 129, 136, 139
ATB_ALLOCFLAGS
constant ... 127, 129, 139
ATB_CACHEFLAGS
constant ... 127, 129, 139
ATB_CACHENOREUSE
constant ... 128, 139
ATB_CACHERANDOM
constant ... 128, 139
ATB_CACHESEQUENTIAL
constant ... 128, 139
ATB_DIOENABLE
constant ... 128, 139

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

ATB_DIOFLAGS
constant ... 127, 129, 140
ATB_DIOSIZE
constant ... 127, 129, 140
ATB_SEQ
constant ... 127, 140
ATC_ALLOC
constant ... 136, 140
ATC_ALLOCINCR
constant ... 135, 140
ATC_ATINCR
constant ... 135, 140
ATC_BIOINCR
constant ... 135, 140
ATC_CACHENOREUSE
constant ... 128, 140
ATC_CACHERANDOM
constant ... 128, 140
ATC_CACHESEQUENTIAL
constant ... 128, 140
ATC_DIOCAPABLE
constant ... 128, 140
ATC_DIOINCR
constant ... 135, 140
ATC_DIOREQUIRED
constant ... 128, 140
ATC_EXPONENT
constant ... 128, 140
ATC_LIST
constant ... 128, 140
ATC_MULTIPLY
constant ... 128, 140
ATC_SEQ
constant ... 127, 140
atomic
pragma ... 84, 140
Atomic_IO_Capabilities
type ... 128, 140
Atomic_IO_Increment_List
enumeration literal ... 135, 140
Attributes
type ... 127, 140
Await_IO
procedure ... 62, 140
Await_Signal
procedure ... 40, 140
Await_Signal_Or_Timeout
procedure ... 30, 40, 140

B

Bad_File_Descriptor
constant ... 15, 140
Bad_Message
constant ... 23, 140
Blocking_Behavior
type ... 21, 140
Broken_Pipe
constant ... 15, 140
Buffered_IO_Capabilities
type ... 129, 140
Buffered_IO_Increment_List
enumeration literal ... 135, 140

C

Cache_Capabilities
type ... 128–129, 140
Cache_No_Reuse
constant ... 128, 140
Cache_Options
type ... 127–128, 140
Cache_Options_Component
constant ... 127, 140
Cache_Random_Access
constant ... 128, 140
Cache_Sequential_Access
constant ... 128, 140
Calendar
package ... 25–26, 30, 105–106, 140
Cancel
function ... 61, 140
Cancelation_Status
type ... 56, 61, 140
Canceled
enumeration literal ... 61, 140
Change_Permissions
procedure ... 50, 140
Change_Protection
procedure ... 80, 88, 140
Characters_Transferred
function ... 60, 64, 140
Clock
function ... 105–106, 140
clock_getres
function ... 101, 140
clock_gettime
function ... 101, 140
Clock_ID
type ... 100, 140
clockid_t
type ... 100, 140
Clock_Realtime
constant ... 100, 102, 106, 140
CLOCK_REALTIME
constant ... 100, 140
clock_settime
function ... 101, 140
Close
procedure ... 70, 81, 113, 140
close
function ... 81, 140
Component_ID_Set
type ... 126–127, 130, 140
Constraint_Error

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

exception ... 25–26, 28, 57–63, 101, 103, 106, 114–115, 140
Control_Block
type ... 56, 141
Control_Block_List
type ... 65, 141
Control_Block_Pointer
type ... 56, 141
Count_List
type ... 128, 141
Create_Realtime_File
procedure ... 131, 141
Create_Timer
function ... 102, 141
Current_Pages
constant ... 76, 141

D

Data_Synchronized
constant ... 53, 141
{DELAYTIMER_MAX}
limit ... 20, 141
Delete_Timer
procedure ... 102, 141
Destroy
procedure ... 68, 141
Direct_IO_Capabilities
type ... 128–129, 141
Direct_IO_Increment_List
enumeration literal ... 135, 141
Direct_IO_Is_Possible
constant ... 128, 141
Direct_IO_Is_Required
constant ... 128, 141
Direct_IO_Options
type ... 127–128, 141
Direct_IO_Options_Component
constant ... 127, 141
Direct_IO_Transfer_Component
constant ... 127, 141
Directory_Not_Empty
constant ... 15, 141
Done
enumeration literal ... 61, 141
Duration
type ... 25, 30, 33, 105, 141

E

[E2BIG]
error status value ... 15, 141
[EACCES]
error status value ... 15, 141
[EAGAIN]
error status value ... 15, 71–72, 141
[EBADF]
error status value ... 15, 141
[EBADMSG]
error status value ... 23, 141
[EBUSY]
error status value ... 15, 141
[ECANCELED]
error status value ... 23, 30, 141
[ECHILD]
error status value ... 15, 141
[EDEADLK]
error status value ... 15, 141
[EEXIST]
error status value ... 15, 141
[EFBIG]
error status value ... 15, 141
[EINPROGRESS]
error status value ... 23, 141
[EINTR]
error status value ... 15, 141
[EINVAL]
error status value ... 106, 141
[EINVAL]
error status value ... 15, 141
[EIO]
error status value ... 15, 141
[EISDIR]
error status value ... 15, 141
[EMFILE]
error status value ... 15, 141
[EMLINK]
error status value ... 15, 141
[EMSGSIZE]
error status value ... 23, 141
[ENAMETOOLONG]
error status value ... 15, 141
[ENFILE]
error status value ... 15, 141
[ENODEV]
error status value ... 15, 141
[ENOENT]
error status value ... 15, 141
[ENOEXEC]
error status value ... 15, 141
[ENOLCK]
error status value ... 15, 141
[ENOMEM]
error status value ... 15, 141
[ENOSPC]
error status value ... 15, 141
[ENOSYS]
error status value ... 15, 29, 141
[ENOTDIR]
error status value ... 15, 141
[ENOTEMPTY]
error status value ... 15, 141
[ENOTSUP]
error status value ... 23, 29, 141
[ENOTTY]
error status value ... 15, 141

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

[ENXIO]
error status value ... 15, 142
[EPERM]
error status value ... 15, 142
[EPIPE]
error status value ... 15, 142
[EROFS]
error status value ... 15, 142
errno
variable ... 14, 29, 106, 142
Error_Code
type ... 14, 142
[ESPIPE]
error status value ... 15, 142
[ESRCH]
error status value ... 15, 142
[EXDEV]
error status value ... 15, 142
Exec
procedure ... 84, 90, 142
Exec_Format_Error
constant ... 15, 142

F

fchmod
function ... 50, 142
fdatasync
function ... 17, 54, 142
FIFO_Within_Priority
constant ... 92, 142
File_Exists
constant ... 15, 142
Filename_Too_Long
constant ... 15, 142
File_Synchronized
constant ... 53, 142
File_Sync_Is_Supported
function ... 47, 142
File_Sync_Support
subtype ... 18, 142
File_Too_Large
constant ... 15, 142
Fork
procedure ... 84, 90, 142
fpathconfig
function ... 52, 142
From_Async_IO
constant ... 38, 142
From_Message_Queue
constant ... 38, 142
From_Queue_Signal
constant ... 38, 142
From_Timer
constant ... 38, 142
From_User
constant ... 38, 142
fsync
function ... 17, 52, 54, 142
ftruncate
function ... 51, 81, 142
Future_Pages
constant ... 76, 142

G

Generic_Receive
generic unit ... 115, 142
Get_Allocation_Increment_List
procedure ... 136, 142
Get_Allocation_Options
function ... 128, 142
Get_Allocation_Size
function ... 128, 142
Get_Attributes
procedure ... 117, 131, 142
Get_Buffer
function ... 57, 142
Get_Buffered_IO_Seek_Boundary
function ... 136, 142
Get_Buffered_IO_Transfer_Seek_Align-
ment_Boundary
function ... 136, 142
Get_Cache_Options
function ... 128–129, 142
Get_Capabilities
procedure ... 132–134, 142
Get_Direct_IO_Options
function ... 128, 142
Get_Direct_IO_Transfer
function ... 128, 142
Get_Error_Code
function ... 14, 59–60, 64, 142
Get_Error_Code
procedure ... 59, 142
Get_File
function ... 57, 142
Get_File_Control
function ... 53, 142
Get_Increment
function ... 129–130, 142
Get_Increment_List
procedure ... 136, 142
Get_Increment_Mode
function ... 129–130, 142
Get_IO_Operation
function ... 57, 142
Get_Length
function ... 57, 142
Get_Max_Messages
function ... 111, 142
Get_Max_Priority
function ... 96–97, 142
Get_Max_Size
function ... 129, 142
Get_Max_Transfer

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

function ... 129–130, 142
Get_Memory_Alignment
function ... 130, 143
Get_Message_Count
function ... 111, 143
Get_Message_Length
function ... 111, 143
Get_Min_Priority
function ... 96–97, 143
Get_Min_Size
function ... 129, 143
Get_Min_Transfer
function ... 129–130, 143
Get_Nanoseconds
function ... 24, 143
Get_Offset
function ... 57, 143
Get_Options
function ... 111, 129–130, 143
Get_Priority
function ... 57, 64, 92, 143
Get_Receiver_Count
function ... 111, 143
Get_Resolution
function ... 101, 143
Get_Round_Robin_Interval
function ... 96, 143
Get_Scheduling_Parameters
function ... 94, 143
Get_Scheduling_Policy
function ... 95, 143
Get_Seconds
function ... 24, 143
Get_Seek_Alignment
function ... 129–130, 143
Get_Seek_Boundary
function ... 130, 143
Get_Sender_Count
function ... 111, 143
Get_Signal
function ... 37–39, 143
Get_Signal_Event
function ... 57, 143
Get_Source
function ... 38–39, 143
Get_Time
function ... 101, 106, 143
Get_Times
function ... 104, 143
Get_Value
function ... 37–39, 72–73, 143

H

Handler_Type
type ... 43, 143
Has_Value
function ... 39, 143

I

Image
function ... 27, 143
Improper_Link
constant ... 15, 143
Inappropriate_IO_Control_Operation
constant ... 15, 143
Increment_By_Exponents
enumeration literal ... 128, 143
Increment_By_List
enumeration literal ... 128, 143
Increment_By_Multiples
enumeration literal ... 128, 143
Increment_List_ID
type ... 135, 143
Increment_Mode
type ... 127–128, 143
Initialize
procedure ... 68, 73, 143
Input_Output_Error
constant ... 15, 143
int
type ... 36, 38, 56, 76, 78, 80, 92, 127–128, 143
Integer
type ... 10, 143
Interrupted_Operation
constant ... 15, 107, 143
Interrupt_Management
package ... 43, 143
{INTERVAL_MAX}
limit ... 19, 46, 143
Invalid_Argument
constant ... 15, 37, 88, 102, 143
Invalid_Seek
constant ... 15, 143
IO_Buffer_Pointer
type ... 56, 64, 143
IO_Offset
type ... 56, 127–128, 143
Is_A_Directory
constant ... 15, 143
Is_Message_Queue
procedure ... 50, 143
Is_Realtime_File
procedure ... 50, 143
Is_Semaphore
procedure ... 50, 143
Is_Shared_Memory
procedure ... 50, 143
Itimerspec
type ... 105, 143
itimerspec
type ... 99, 105, 143

J

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

K

L

Last
function ... 64, 144
length
attribute ... 64, 144
lio_listio
function ... 59, 144
LIO_NOP
constant ... 56, 144
LIO_NOWAIT
constant ... 59, 144
LIO_READ
constant ... 56, 144
LIO_WAIT
constant ... 59, 144
LIO_WRITE
constant ... 56, 144
List_IO_Max
function ... 47, 144
List_IO_Maxima
subtype ... 20, 144
List_IO_No_Wait
procedure ... 59, 144
List_IO_Operation
type ... 56, 144
List_IO_Operations
type ... 56, 144
List_IO_Wait
procedure ... 59, 144
Lock_All
procedure ... 76, 89, 144
Lock_In_Memory
generic unit ... 83, 144
Lock_In_Memory
procedure ... 89, 144
Lock_Range
procedure ... 28, 77, 144
long
type ... 23, 144
longjmp
function ... 42, 107, 144

M

MAP_FIXED
constant ... 79, 144
Map_Memory
function ... 85, 144
Map_Memory
procedure ... 28, 79, 144
Mapped_Files_Are_Supported
function ... 47, 144
Mapped_Files_Support
subtype ... 18, 144
Mapped_Memory_Region
type ... 88, 144
MAP_SHARED
constant ... 79, 144
Masked_Signals
procedure ... 71, 144
Mask_Signals
pragma ... 44, 144
MCL_CURRENT
constant ... 76, 144
MCL_FUTURE
constant ... 76, 144
Memory_Locking_Is_Supported
function ... 47, 144
Memory_Locking_Options
type ... 76, 144
Memory_Locking_Support
subtype ... 18, 144
Memory_Protection_Is_Supported
function ... 47, 144
Memory_Protection_Support
subtype ... 18, 144
Memory_Range_Locking_Is_Supported
function ... 47, 144
Memory_Range_Locking_Support
subtype ... 18, 144
Message_Passing_Is_Supported
function ... 47, 144
Message_Passing_Support
subtype ... 18, 144
Message_Priority
type ... 118, 144
Message_Queue_Attributes
type ... 110, 144
Message_Queue_Blocking_Behavior
subtype ... 21, 144
Message_Queue_Descriptor
type ... 110, 144
Message_Queue_Non_Blocking
constant ... 110, 144
Message_Queue_Open_Max
function ... 47, 144
Message_Queue_Open_Maxima
subtype ... 20, 144
Message_Queue_Option_Set
type ... 117, 144
Message_Queue_Priority_Max
function ... 47, 144
Message_Queue_Priority_Maxima
subtype ... 20, 144
Message_Too_Long
constant ... 23, 144
mlock
function ... 77, 88, 144
mlockall
function ... 76, 88, 144
mmap
function ... 27, 29, 78–79, 87–88, 144

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

mode_t
type ... 110, 145
mprotect
function ... 79–80, 89, 145
mq_attr
type ... 110, 145
mq_close
function ... 113, 145
mq_destroy
function ... 113, 145
mqd_t
type ... 110, 145
mq_getattr
function ... 117, 145
MQ_NONBLOCK
constant ... 110, 145
mq_notify
function ... 116, 145
mq_open
function ... 111–112, 117, 145
{MQ_OPEN_MAX}
limit ... 20, 145
{MQ_PRIO_MAX}
limit ... 20, 145
mq_receive
function ... 115, 145
mq_send
function ... 114, 145
mq_setattr
function ... 116–117, 145
mq_unlink
function ... 113, 145
MS_ASYNC
constant ... 80, 145
MS_INVALIDATE
constant ... 80, 145
MS_SYNC
constant ... 80, 145
msync
function ... 80, 88, 145
munlock
function ... 77, 145
munlockall
function ... 76, 145
munmap
function ... 29, 79, 145

N

Nanoseconds
subtype ... 23, 30–31, 145
Nanoseconds_Base
type ... 23, 30–31, 145
Nanosleep
procedure ... 104–107, 145
nanosleep
function ... 104, 106–107, 145
No_Child_Process
constant ... 15, 145
No_Error
constant ... 23, 29, 59, 145
No_Locks_Available
constant ... 15, 145
No_Op
enumeration literal ... 56, 145
No_Space_Left_On_Device
constant ... 15, 145
No_Such_Device_Or_Address
constant ... 15, 145
No_Such_File_Or_Directory
constant ... 15, 145
No_Such_Operation_On_Device
constant ... 15, 88, 145
No_Such_Process
constant ... 15, 145
Not_A_Directory
constant ... 15, 145
Not_Canceled
enumeration literal ... 61, 145
Not_Enough_Space
constant ... 15, 89, 145
NULL
constant ... 61–62, 102, 112, 116, 145
Null_Address
constant ... 27, 78–79, 85, 145
Null_Process_ID
constant ... 41, 145

O

O_CREAT
constant ... 69, 112, 145
O_DSYNC
constant ... 11, 53, 63–64, 145
O_EXCL
constant ... 82, 112, 145
off_t
type ... 56, 127–128, 145
O_NONBLOCK
constant ... 112, 145
Open
function ... 53, 69, 87–88, 90, 107, 112, 145
open
function ... 11, 72, 145
Open_Mapped_Memory_Region
procedure ... 87, 145
Open_Option_Set
type ... 22, 53, 145
Open_Or_Create
function ... 53, 69, 73, 90, 112, 145
Open_Or_Create_Shared_Memory
function ... 82, 145
Open_Shared_Memory
function ... 82, 87–88, 90, 145
Operation_Canceled
constant ... 23, 145

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Operation_In_Progress
constant ... 23, 146
Operation_Not_Implemented
constant ... 15, 88–89, 146
Operation_Not_Permitted
constant ... 15, 88, 146
Operation_Not_Supported
constant ... 23, 88, 146
Option_Set
type ... 22, 146
O_RDONLY
constant ... 82, 112, 146
O_RDWR
constant ... 82, 112, 146
O_SYNC
constant ... 11, 53, 63–64, 146
Other_Scheduling
constant ... 92, 146
O_TRUNC
constant ... 82, 146
O_WRONLY
constant ... 112, 146

P

Page_Size
function ... 28, 47, 146
{PAGESIZE}
limit ... 20, 31, 146
Page_Size_Range
subtype ... 19–20, 146
pathconfig
function ... 52, 146
_PC_ASYNC_IO
constant ... 52, 146
_PC_PRIO_IO
constant ... 52, 146
_PC_SYNC_IO
constant ... 52, 146
Permission_Denied
constant ... 15, 88, 146
Permission_Set
type ... 110, 146
Portable_Async_IO_Max
constant ... 19, 146
Portable_Clock_Resolution_Min
constant ... 19, 31, 146
Portable_List_IO_Max
constant ... 19, 146
Portable_Message_Queue_Open_Max
constant ... 19, 146
Portable_Message_Queue_Priority_Max
constant ... 19, 146
Portable_Queued_Signals_Max
constant ... 19, 146
Portable_Realtime_Signal_Max
subtype ... 36, 146
Portable_Realtime_Signals_Max
constant ... 19, 146
Portable_Semaphores_Max
constant ... 19–20, 146
Portable_Semaphores_Value_Max
constant ... 19, 146
Portable_Timer_Overruns_Max
constant ... 19, 146
Portable_Timers_Max
constant ... 19, 146
POSIX
package ... 16, 22, 25, 29–30, 32, 36, 56, 100, 128, 146
POSIX_Address_Arithmetic
package ... 32, 78, 85, 146
{_POSIX_AIO_LISTIO_MAX}
limit ... 19, 146
{_POSIX_AIO_MAX}
limit ... 19, 146
POSIX_Asynchronous_IO
package ... 17, 21, 55, 146
{_POSIX_ASYNCHRONOUS_IO}
limit ... 17–18, 146
POSIX_Asynchronous_IO.Get_Error_Code
function ... 23, 146
{_POSIX_CLOCKRES_MIN}
limit ... 19, 146
POSIX_Configurable_File_Limits
package ... 51–52, 146
POSIX_Configurable_System_Limits
package ... 28, 45, 48, 96, 146
{_POSIX_DELAYTIMER_MAX}
limit ... 19, 146
POSIX_Error
exception ... 6, 14, 31, 37, 60, 64, 88–89, 101–102, 107, 146
POSIX_Files
package ... 49, 52, 146
POSIX_File_Status
package ... 49, 146
{_POSIX_FSYNC}
limit ... 17–18, 146
POSIX_IO
package ... 9, 17, 52–53, 63, 81, 127, 146
{_POSIX_MAPPED_FILES}
limit ... 17–18, 146
{_POSIX_MEMLOCK}
limit ... 17–18, 146
{_POSIX_MEMLOCK_RANGE}
limit ... 17–18, 146
POSIX_Memory_Locking
package ... 17, 75, 146
POSIX_Memory_Mapping
package ... 17, 77, 146
{_POSIX_MEMORY_PROTECTION}
limit ... 17–18, 88, 146
POSIX_Memory_Range_Locking
package ... 17, 75, 146
POSIX_Message_Passing

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

package ... 17, 109, 146
{_POSIX_MESSAGE_PASSING}
limit ... 17–18, 147
POSIX_Message_Queues
package ... 21, 147
{_POSIX_MQ_OPEN_MAX}
limit ... 19, 147
{_POSIX_MQ_PRIO_MAX}
limit ... 19, 147
POSIX_Permissions
package ... 110, 147
{_POSIX_PRIORITIZED_IO}
limit ... 17–18, 147
POSIX_Priority_Scheduling
package ... 17, 91, 147
{_POSIX_PRIORITY_SCHEDULING}
limit ... 17–18, 147
POSIX_Realtime_Files
package ... 17, 21, 119, 147
{_POSIX_REALTIME_FILES}
limit ... 17–18, 147
{_POSIX_REALTIME_SIGNALS}
limit ... 17–18, 147
{_POSIX_RTSIG_MAX}
limit ... 19, 147
POSIX_Semaphores
package ... 17, 21, 67, 147
{_POSIX_SEMAPHORES}
limit ... 17–18, 147
{_POSIX_SEM_NSEMS_MAX}
limit ... 19, 147
{_POSIX_SEM_VALUE_MAX}
limit ... 19, 147
POSIX_Shared_Memory_Objects
package ... 17, 21, 81, 147
{_POSIX_SHARED_MEMORY_OBJECTS}
limit ... 17–18, 147
POSIX_Signal
type ... 36, 147
POSIX_Signals
package ... 17, 21, 30, 35–36, 39, 56, 147
{_POSIX_SIGQUEUE_MAX}
limit ... 19, 147
POSIX_Synchronized_IO
package ... 64, 147
{_POSIX_SYNCHRONIZED_IO}
limit ... 17–18, 147
POSIX_Time_Conversions
package ... 25, 147
{_POSIX_TIMER_MAX}
limit ... 19, 147
POSIX_Timers
package ... 17, 21, 30, 99, 147
{_POSIX_TIMERS}
limit ... 17–18, 147
Post
procedure ... 71, 147
Prioritized_IO_Is_Supported
function ... 47, 52, 147
Prioritized_IO_Support
subtype ... 18, 147
Priority
type ... 64, 147
Priority_Scheduling_Is_Supported
function ... 47, 147
Priority_Scheduling_Support
subtype ... 18, 147
Process_ID
type ... 9, 147
Program_Error
exception ... 88–89, 147
Protection_Options
type ... 78, 147
PROT_EXEC
constant ... 78, 147
PROT_NONE
constant ... 78, 147
PROT_READ
constant ... 78, 147
PROT_WRITE
constant ... 78, 147
pthread_cond_timed_wait
function ... 107
pthread_cond_timedwait
function ... 108
pthread_cond_timed_wait
function ... 147
pthread_cond_timedwait
function ... 147

Q

Queued_Signals_Max
function ... 47, 147
Queued_Signals_Maxima
subtype ... 20, 147
Queue_Signal
procedure ... 41, 147

R

Read
enumeration literal ... 56, 147
Read
procedure ... 57, 147
Read_Only
enumeration literal ... 82, 147
Read_Only_File_System
constant ... 15, 147
Read_Write
enumeration literal ... 82, 147
Realtime_File_Blocking_Behavior
subtype ... 21, 147
Realtime_Files_Are_Supported
function ... 47, 147
Realtime_Files_Support

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

subtype ... 18, 147
Realtime_Signal
subtype ... 36, 148
Realtime_Signal_Blocking_Behavior
subtype ... 21, 148
Realtime_Signals_Are_Supported
function ... 47, 148
Realtime_Signals_Max
function ... 47, 148
Realtime_Signals_Maxima
subtype ... 20, 148
Realtime_Signals_Support
subtype ... 18, 148
Receive
procedure ... 115, 117–118, 148
Remove_Notify
procedure ... 116, 148
Request_Notify
procedure ... 116, 148
Resource_Busy
constant ... 15, 88, 148
Resource_Deadlock_Avoided
constant ... 15, 148
Resource_Temporarily_Unavailable
constant ... 15, 88–89, 148
rf_attrbuf
type ... 127, 129, 135, 148
rf_capaiobuf
type ... 128–129, 148
rf_capallocbuf
type ... 127–129, 148
rf_capbiobuf
type ... 129, 148
rf_capcachcebuf
type ... 129, 148
rf_capcachebuf
type ... 128–129, 148
rf_capdiobuf
type ... 128–129, 148
rf_create
function ... 131, 148
rf_freebuf
function ... 136, 148
rf_getaiocap
function ... 134, 148
rf_getalloccap
function ... 132, 148
rf_getallocincr
function ... 136, 148
rf_getattr
function ... 131, 148
rf_getbiocap
function ... 133, 148
rf_getbuf
function ... 136, 148
rf_getcachecap
function ... 133, 148
rf_getdiocap
function ... 134, 148
rf_getincr
function ... 135–136, 148
rf_setattr
function ... 132, 148
Round_Robin_Within_Priority
constant ... 92, 148
{RTSIG_MAX}
limit ... 20, 148

S

SA_SIGINFO
constant ... 39, 148
{_SC_AIO_LISTIO_MAX}
limit ... 47, 148
{_SC_AIO_MAX}
limit ... 47, 148
{_SC_AIO_PRIO_DELTA_MAX}
limit ... 47, 148
{_SC_ASYNCHRONOUS_IO}
limit ... 47, 148
{_SC_DELAYTIMER_MAX}
limit ... 47, 148
{_SC_FSYNC}
limit ... 47, 148
SCHED_FIFO
constant ... 92, 97, 148
sched_getparam
function ... 94, 148
sched_get_priority_max
function ... 96, 148
sched_get_priority_min
function ... 96, 148
sched_get_rr_interval
function ... 96, 148
sched_getscheduler
function ... 95, 148
SCHED_OTHER
constant ... 92, 148
sched_param
type ... 91–92, 148
SCHED_RR
constant ... 92, 148
sched_rr_get_interval
function ... 96, 148
sched_setparam
function ... 93, 148
sched_setscheduler
function ... 94, 148
Scheduling_Parameters
type ... 92, 148
Scheduling_Policy
type ... 92, 148
Scheduling_Priority
type ... 92, 148
sched_yield
function ... 95, 148

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

{_SC_MAPPED_FILES}
limit ... 47, 149
{_SC_MEMLOCK}
limit ... 47, 149
{_SC_MEMLOCK_RANGE}
limit ... 47, 149
{_SC_MEMORY_PROTECTION}
limit ... 47, 149
{_SC_MESSAGE_PASSING}
limit ... 47, 149
{_SC_MQ_OPENMAX}
limit ... 47, 149
{_SC_MQ_PRIO_MAX}
limit ... 47, 149
{_SC_PAGESIZE}
limit ... 20, 47, 149
{_SC_PRIORITIZED_IO}
limit ... 47, 149
{_SC_PRIORITY_SCHEDULING}
limit ... 47, 149
{_SC_REALTIME_FILES}
limit ... 47, 149
{_SC_REALTIME_SIGNALS}
limit ... 47, 149
{_SC_RTSIG_MAX}
limit ... 47, 149
{_SC_SEMAPHORES}
limit ... 47, 149
{_SC_SEM_NSEMS_MAX}
limit ... 47, 149
{_SC_SEM_VALUE_MAX}
limit ... 47, 149
{_SC_SHARED_MEMORY_OBJECTS}
limit ... 47, 149
{_SC_SIGQUEUE_MAX}
limit ... 47, 149
{_SC_SYNCHRONIZED_IO}
limit ... 47, 149
{_SC_TIMER_MAX}
limit ... 47, 149
{_SC_TIMERS}
limit ... 47, 149
Seconds
type ... 23, 149
Semaphore
type ... 67, 149
Semaphore_Blocking_Behavior
subtype ... 21, 149
Semaphore_Is_Supported
function ... 47, 149
Semaphores_Max
function ... 20, 47, 149
Semaphores_Maxima
subtype ... 20, 149
Semaphore_Support
subtype ... 18, 149
Semaphores_Value_Max
function ... 47, 149
Semaphores_Value_Maxima
subtype ... 20, 149
sem_close
function ... 70, 149
sem_destroy
function ... 68–69, 149
sem_get_value
function ... 72–73, 149
sem_init
function ... 68, 72–73, 149
{SEM_NSEMS_MAX}
limit ... 20, 149
sem_open
function ... 69, 72–73, 149
sem_post
function ... 71, 149
sem_t
type ... 67–68, 149
sem_trywait
function ... 71, 149
sem_unlink
function ... 70, 149
{SEM_VALUE_MAX}
limit ... 20, 149
sem_wait
function ... 71, 149
Send
procedure ... 114, 117, 149
Sequential_Allocation
constant ... 127, 149
Set_Absolute
procedure ... 104, 149
Set_Allocation_Options
procedure ... 128, 149
Set_Allocation_Size
procedure ... 128, 149
Set_Attributes
procedure ... 117, 132, 149
Set_Buffer
procedure ... 57, 149
Set_Cache_Options
procedure ... 128–129, 149
Set_Direct_IO_Options
procedure ... 128, 149
Set_Direct_IO_Transfer
procedure ... 128, 149
Set_File
procedure ... 57, 149
Set_File_Control
procedure ... 53, 149
Set_Increment
procedure ... 129–130, 149
Set_Increment_Mode
procedure ... 129–130, 149
Set_IO_Operation
procedure ... 57, 149
setjmp
function ... 107, 149

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Set_Length
procedure ... 57, 150
Set_Max_Messages
procedure ... 111, 150
Set_Max_Size
procedure ... 129, 150
Set_Max_Transfer
procedure ... 129–130, 150
Set_Memory_Alignment
procedure ... 130, 150
Set_Message_Length
procedure ... 111, 150
Set_Min_Size
procedure ... 129, 150
Set_Min_Transfer
procedure ... 129–130, 150
Set_Nanoseconds
procedure ... 24, 150
Set_Offset
procedure ... 57, 150
Set_Options
procedure ... 111, 129–130, 150
Set_Overruns
function ... 104, 150
Set_Priority
procedure ... 57, 64, 92, 150
Set_Relative
procedure ... 104, 150
Set_Scheduling_Parameters
procedure ... 93, 150
Set_Scheduling_Policy
procedure ... 94, 150
Set_Seconds
procedure ... 24, 150
Set_Seek_Alignment
procedure ... 129–130, 150
Set_Seek_Boundary
procedure ... 130, 150
Set_Signal
procedure ... 37–39, 150
Set_Signal_Event
procedure ... 57, 150
Set_Source
procedure ... 38–39, 150
Set_Time
procedure ... 101, 106, 150
Set_Value
procedure ... 37–39, 150
Shared_Memory_Blocking_Behavior
subtype ... 21, 150
Shared_Memory_Objects_Are_Supported
function ... 47, 150
Shared_Memory_Objects_Support
subtype ... 18, 150
Sheduling_Parameters
type ... 91, 150
shm_open
function ... 81–82, 87, 150
shm_unlink
function ... 82, 150
SI_ASYNCIO
constant ... 38–39, 150
sigaction
function ... 39, 150
SIGALRM
constant ... 42, 108, 150
SIGBUS
constant ... 36, 150
SIGBUS
constant ... 36–37, 150
sigevent
type ... 36–37, 41, 56, 150
siginfo_t
type ... 38–39, 150
Signal
package ... 38, 150
Signal
type ... 36, 150
Signal_Alarm
constant ... 102, 105, 150
Signal_Event
type ... 36–37, 56, 150
Signal_Info
type ... 38, 150
Signal_Reference
function ... 36–37, 150
Signal_Source
type ... 38, 150
Signal_Value
type ... 36, 150
sigqueue
function ... 41, 150
{SIGQUEUE_MAX}
limit ... 20, 150
{SIGRTMAX}
limit ... 36–37, 150
{SIGRTMIN}
limit ... 36–37, 150
sigtimedwait
function ... 40–41, 150
sigval
type ... 36, 41, 150
sigwaitrt
function ... 40–41, 150
SI_MESGQ
constant ... 38–39, 150
SI_QUEUE
constant ... 38–39, 150
SI_TIMER
constant ... 38–39, 150
SI_USER
constant ... 38–39, 150
size
attribute ... 83, 150
size_t
type ... 56, 110, 128, 150

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

sleep
function ... 106, 151
small
attribute ... 31, 151
Split
procedure ... 25–26, 151
Standard
package ... 25, 151
Start_Process
procedure ... 84, 90, 151
Storage_Count
type ... 90, 151
Storage_Offset
type ... 90, 151
S_TYPEISMQ
macro ... 50, 151
S_TYPEISRTFILE
macro ... 50, 151
S_TYPEISSEM
macro ... 50, 151
S_TYPEISSHM
macro ... 50, 151
Synchronization_Options
type ... 80, 151
Synchronize_Data
procedure ... 54, 63, 151
Synchronize_File
procedure ... 54, 63, 151
Synchronize_Memory
procedure ... 80, 88, 151
Sync_IO_Is_Supported
function ... 47, 52, 151
Sync_IO_Support
subtype ... 18, 151
sysconfig
function ... 31, 151
System
package ... 27, 32, 37, 64, 84–85, 90, 151

T

Tasks
... 29, 151
Time
type ... 25–26, 30, 106, 151
TIMER_ABSTIME
constant ... 104, 151
Timer_Blocking_Behavior
subtype ... 21, 151
timer_create
function ... 102, 151
timer_delete
function ... 102–103, 151
timer_getoverrun
function ... 103–104, 151
timer_gettime
function ... 103–104, 151
Timer_ID
type ... 100, 151
{TIMER_MAX}
limit ... 20, 102, 151
Timer_Overruns_Max
function ... 47, 151
Timer_Overruns_Maxima
subtype ... 20, 151
Timers_Are_Supported
function ... 47, 151
timer_settime
function ... 103–104, 151
Timers_Max
function ... 20, 47, 151
Timers_Maxima
subtype ... 20, 151
Timers_Support
subtype ... 18, 151
timer_t
type ... 100, 151
Timespec
type ... 24–26, 30, 100–101, 151
timespec
type ... 23–24, 30, 100, 151
Timespec_Of
function ... 25–26, 151
time_t
type ... 23–24, 151
To_Calendar_Time
function ... 25–26, 151
To_Duration
function ... 25–26, 151
Too_Many_Links
constant ... 15, 151
Too_Many_Open_Files
constant ... 15, 151
Too_Many_Open_Files_In_System
constant ... 15, 151
To_Timespec
function ... 25–26, 151
Truncate_File
procedure ... 51, 151
Try_Wait
procedure ... 71–72, 151

U

UNCHECKED_CONVERSION
generic unit ... 11
Unchecked_Conversion
generic unit ... 37
UNCHECKED_CONVERSION
generic unit ... 151
Unchecked_Conversion
generic unit ... 151
Unlink
procedure ... 70, 113, 151
Unlink_Shared_Memory
procedure ... 82, 151

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

`Unlock_All`
procedure ... 76, 152
`Unlock_Range`
procedure ... 28, 77, 152
`Unmap_Memory`
procedure ... 28, 79, 152
unsigned
type ... 127, 152

V

`Value`
function ... 27, 152
`volatile`
pragma ... 84–85, 152

W

`Wait`
procedure ... 71–72, 152
`Write`
enumeration literal ... 56, 152
`Write`
procedure ... 58, 152

X

Y

`Yield`
procedure ... 95, 152

Z

Copyright ©1992 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.