

IEEE Standard for Information Technology—POSIX FORTRAN 77 Language Interfaces—Part 1: Binding for System Application Program Interface (API)

Sponsor

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

Approved June 18, 1992

IEEE Standards Board

Abstract: This standard provides a standardized interface for accessing the system services of ISO/IEC 9945-1: 1990 (IEEE Std 1003.1-1990, also known as POSIX.1), and support routines to access constructs not directly accessible with FORTRAN 77. This standard supports application portability at the source level through the binding between ANSI X3.9-1978 and POSIX.1, and a standardized definition of language-specific services. The goal is to provide standardized interfaces to the POSIX.1 system services via a FORTRAN 77 language interface. Terminology and general requirements, process primitives, the process environment, files and directories, input and output primitives, device- and class-specific functions, the FORTRAN 77 language library, and system databases are covered.

Keywords: application portability, FORTRAN 77, interfaces, interoperability, POSIX, system interfaces

The Institute of Electrical and Electronics Engineers, Inc.

345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1992 by the Institute of Electrical and Electronics Engineers, Inc.

All rights reserved. Published 1992 Printed in the United States of America

ISBN 1-55937-230-3

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

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

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

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

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

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

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

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

Introduction

(This introduction is not a normative part of IEEE Std 1003.9-1992, IEEE Standard for Information Technology—POSIX FORTRAN 77 Language Interfaces—Part 1: Binding for System Application Program Interface (API), but is included for information only.)

The purpose of this standard is to support application portability at the source level through the definition of:

- 1) An interface between the ANSI X3.9-1978 FORTRAN Standard (archival) and ISO/IEC 9945-1: 1990, Information technology — Portable Operating System Interface (POSIX) — Part 1: System application interface (API) [C language]
- 2) A standardized interface for language-specific services.

The focus of this standard is to provide standardized interfaces to the ISO/IEC 9945-1: 1990 system services via a FORTRAN 77 language interface. Future work will consist of interfaces to other parts of ISO/IEC 9945 and the possible use of new functionality provided in ISO/IEC 1539: 1991 (Fortran 90).

Organization of This Standard

- 1) Statement of scope and list of normative references (Section 1)
- 2) Definitions and global concepts (Section 2)
- 3) The various interface facilities (Section 3 through 9)

The FORTRAN 77 language interface for each service interface is given in the subclause labeled Synopsis. The correspondence of the ISO/IEC 9945-1: 1990 system service interface to the FORTRAN 77 language interface is described in the Description subclause. Additional information on the creation of specific actual arguments is provided for some interfaces. The Description subclause provides a specification of the operation performed for the language-specific services. In most cases, there is also an Errors subclause that describes error handling. References are used to direct the reader to related sections in ISO/IEC 9945-1: 1990 and in POSIX.9. Additional material to complement sections in this standard may be found in the Rationale and Notes (Annex A). This annex provides historical perspectives into the technical choices made by developers of this standard. It also elaborates on the information provided in the corresponding section of this standard.

Informative annexes are not part of the draft standard and are provided for information only. A normative annex is part of the standard and imposes requirements, but there are currently no such normative annexes in this standard.

In publishing this standard, its developers simply intend to provide a basis upon which various FORTRAN 77 interfaces to ISO/IEC 9945-1: 1990 can be measured for conformance. It is not the intent of the developers to measure or rate any products, to reward or sanction any vendors of products for conformance or lack of conformance to this standard, or to enforce this standard by these or any other means. The responsibility for determining the degree of conformance or lack thereof with this standard rests solely with the individual who is evaluating the product claiming to be in conformance with this standard.

Background

The developers of this standard represent a cross section of hardware manufacturers, user organizations, software designers, applications programmers, and others. In the course of the development of this standard, the developers received guidance from members of the ANS Committee on Fortran, X3J3.

ISO/IEC 9945-1: 1990 describes a set of fundamental system services. Access to these services has been provided by defining an interface using the FORTRAN 77 programming language in this standard.

Audience

The intended audience for this standard is all persons concerned with an industrywide standard FORTRAN 77 interface to the system services described in ISO/IEC 9945-1: 1990.

Purpose

Several principles guided the development of this standard:

Application Oriented

The basic goal was to promote portability of FORTRAN 77 application programs on systems conforming to ISO/IEC 9945-1: 1990.

Interface, Not Implementation

This standard defines an interface, not an implementation. No details of the implementation of any function are given, although historical practice may be indicated in Annex A.

Source, Not Object, Portability

This standard has been written so that a program written and translated for execution on one conforming implementation may also be translated for execution on another conforming implementation. This standard does not guarantee that executable (object or binary) code will execute under a different conforming implementation than that for which it was translated, even if the underlying hardware is identical.

The FORTRAN 77 Language

This standard is written in terms of the standard FORTRAN 77 language as specified in the FORTRAN 77 standard {3}. See 1.3.3. It contains the single extension of 31-character names.

Minimal Interface, Minimally Defined

In keeping with the rules of FORTRAN 77, this standard uses subroutine and function calls to interface to the ISO/IEC 9945-1: 1990. The 31-character name extension was added to allow the use of interface names from ISO/IEC 9945-1: 1990.

Related Standards Activities

Activities to extend this standard to address additional requirements are being considered.

The following activities are under active consideration at this time, or are expected to become active in the near future:

- 1) ISO 1539: 1991 (Fortran 90) Programming Language binding to a language-independent version of ISO/IEC 9945-1: 1990.
- 2) Fortran binding to Shell and Utilities facilities
- 3) Fortran binding to Realtime facilities
- 4) Fortran binding to Security

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

This standard was prepared by the 1003.9 Working Group, sponsored by the Technical Committee on Operating Systems and Application Environments of the IEEE Computer Society. At the time this standard was approved, the membership of the 1003.9 Working Group was as follows:

Technical Committee on Operating Systems and Application Environments (TCOS)

Chair: Jehan-François Pâris

TCOS Standards Subcommittee

Chair: Jim Isaak

**Vice-Chairs: Ralph Barker
Hal Jespersen
Lorraine Kevra
Pete Meier
Andrew Twigger
Treasurer: Quin Hahn
Secretary: Shane McCarron**

1003.9 Working Group Officials

**Chair: John J. McGrory II
Vice-Chair: Michael Hannah
Editor: Daniel J. Magenheimer
Secretary: Larry Diegel**

Working Group

Joanne Brixius
Loren Buhle

Cynthia M. Cox
Mike Hunter

Joseph King

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

Roger E. Anderson
Bengt Asker
David Athersych
Edward Benson
Jerry Berkman
Keith Bierman
Andy Bihain
James M. Bishop
Andy Cheese
Kilnam Chon
Cynthia M. Cox
Larry Diegel
Ron Elliott
Roger Golliver
E. Howard Green
Robert M. Gross

Mark Guzzi
Charles Hammons
Michael Hannah
Kurt W. Hirschert
Don Huebschman
Michael T. Hunter
Jim Isaak
Hal Jespersen
Jens Kolind
Ip-Beau Phillip Law
Donald Lewine
F. C. Lim
Daniel J. Magenheimer
Roger Martin
Patrick McGehearty
John J. McGrory II

Robert McWhirter
Lyle Meier
Martha Nalebuff
Daniel Nissen
Fred Noz
Paul E. Renaud
Steve M. Rowan
Lorne H. Schachter
Gerhard Schmitt
Leonard Seagren
Richard Seibel
Dan Shia
Andrew D. Tait
Ravi Tavakley
Donn S. Terry
Mark-Rene Uchida

Michael W. Vannier
Stephen Walli
Richard W. Weaver
Frederick N. Webb

Janusz Zalewski
John Zolnowsky
Joanne Brixius (*ANSI X3J3
Liaison*)

Joseph King (*DECUS
Institutional Representative*)

When the IEEE Standards Board approved this standard on June 18, 1992, it had the following membership:

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

Dennis Bodson
Paul L. Borrill
Clyde Camp
Donald C. Fleckenstein
Jay Forster*
David F. Franklin
Ramiro Garcia
Thomas L. Hannan

Donald N. Heirman
Ben C. Johnson
Walter J. Karplus
Ivor N. Knight
Joseph Koepfinger*
Irving Kolodny
D. N. “Jim” Logothetis
Lawrence V. McCall

T. Don Michael*
John L. Rankine
Wallace S. Read
Ronald H. Reimer
Gary S. Robinson
Martin V. Schneider
Terrance R. Whittemore
Donald W. Zipse

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
James Beall

Richard B. Engelman
David E. Soffrin

Stanley Warshaw

Mary Lynne Nielsen, *IEEE Standards Project Editor*

CLAUSE	PAGE
1. General	1
1.1 Scope	1
1.2 Normative References	1
1.3 Conformance	2
2. Terminology and General Requirements	4
2.1 Conventions	4
2.2 Definitions	5
2.3 FORTRAN 77 Language Bindings Concepts	7
2.4 Error Numbers	10
2.5 Primitive System Data Types	10
2.6 Environment Description	10
2.7 FORTRAN 77 Language Definitions	11
2.8 Numerical Limits	11
2.9 Symbolic Constants	11
3. Process Primitives	12
3.1 Process Creation and Execution	12
3.2 Process Termination	13
3.3 Signals	15
3.4 Timer Operations	21
4. Process Environment	23
4.1 Process Identification	23
4.2 User Identification	24
4.3 Process Groups	27
4.4 System Identification	29
4.5 Time	30
4.6 Environment Variables	32
4.7 Terminal Identification	33
4.8 Configurable System Variables	35
5. Files and Directories	36
5.1 Directories	36
5.2 Get Working Directory	38
5.3 General File Creation	39
5.4 Special File Creation	42
5.5 File Removal	44
5.6 File Characteristics	46
5.7 Configurable Pathname Variables	51
6. Input and Output Primitives	52
6.1 Pipes	52
6.2 File Descriptor Manipulation	53
6.3 File Descriptor Deassignment	54
6.4 Input and Output	54
6.5 Control Operations on Files	56

CLAUSE	PAGE
7. Device- and Class-Specific Procedures	58
7.1 General Terminal Interface	58
7.2 General Terminal Interface Control Subroutines.....	61
8. FORTRAN 77 Language Library	64
8.1 FORTRAN 77 Intrinsic.....	64
8.2 System Symbolic Constant Access	64
8.3 Structure Creation and Manipulation	65
8.4 Subroutine-Handle Manipulation.....	69
8.5 External Unit and File Description Interaction	70
8.6 Stream I/O	78
8.7 Bit Field Manipulation	81
8.8 System Date and Time	83
8.9 Command-Line Arguments.....	84
8.10 Character String Procedures.....	85
8.11 Extended Range Integer Manipulation	86
8.12 Process Termination.....	87
9. System Databases.....	87
9.1 System Databases.....	87
9.2 Database Access.....	87
10. Data Interchange Format.....	91
10.1 Archive/interchange File Format	91
Annex A Rationale and Notes (Informative)	92

IEEE Standard for Information Technology—POSIX FORTRAN 77 Language Interfaces—Part 1: Binding for System Application Program Interface (API)

1. General

1.1 Scope

This standard provides the binding between the ANSI X3.9-1978 (FORTRAN 77) programming language and the system services defined in ISO/IEC 9945-1: 1990 (hereinafter referred to as POSIX.1 {2}).

As with the definition of the service interfaces in POSIX.1 {2}, this FORTRAN 77 language binding is defined exclusively at the source code level. The objective is that a Strictly Conforming Application may be developed in FORTRAN 77 and compiled to execute on a POSIX.1 {2} conforming implementation.

It is intended that this FORTRAN 77 language binding may coexist on a system with any other language binding.

The following areas are outside the scope of this standard:

- 1) Extensions to the FORTRAN 77 language, other than the required longer external names.
- 2) Bindings to any system interfaces using new or changed features in the revision to FORTRAN 77 (ISO/IEC 1539: 1991).
- 3) Bindings for system interfaces other than those defined in POSIX.1 {2}.

1.2 Normative References

[1] ISO/IEC 9899:1990, *Information technology—Programming languages—C*.¹

¹ISO/IEC documents can be obtained from the ISO office, 1, rue de Varembe, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse. IEEE documents can be obtained from the Institute of Electrical and Electronic Engineers, Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ, 08855-1331, USA.

[2] ISO/IEC 9945-1: 1990 (IEEE Std 1003.1-1990), *Information technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language]*.

[3] ANSI X3.9-1978, *American National Standard Programming Language FORTRAN*.²

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Requirements

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

- 1) The system shall support all required interfaces defined within this standard. These interfaces shall support the functional behavior described herein.
- 2) The system may provide additional routines or facilities not required by this standard. Nonstandard extensions should be identified as such in the system documentation. Nonstandard extensions, when used, may change the behavior of routines or facilities defined by this standard. The conformance document shall define an environment in which an application can be made to run with the behavior specified by this standard. In no case shall such an environment require modification of a Strictly Conforming POSIX.9 Application.

See POSIX.1 {2} 1.3 for a description of a POSIX.1 {2} conforming implementation.

1.3.1.2 Documentation

A conformance document with the information described in POSIX.1 {2} 1.3 and in this standard shall be available for an implementation claiming conformance to POSIX.1 {2} and to this standard. The conformance document shall have the same structure as that described in POSIX.1 {2} 1.3 and this standard, with the information presented in appropriately-numbered sections. The conformance document shall not contain information about extended facilities or capabilities outside the scope of POSIX.1 {2} 1.3, this standard, and the applicable standard described in 1.3.3.1.

The conformance document shall contain a statement that indicates the full name, number, and date of the standards that apply. The conformance document may also list international software standards that are available for use by a Conforming POSIX.1 {2} and POSIX.9 Application. Applicable characteristics where documentation is required by one of these standards, or by standards of government bodies, may also be included.

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

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

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

See POSIX.1 {2} 1.3 for a description of a POSIX.1 {2} conformance document.

²This is an archival standard which is identical to the obsolete ISO 1539:1980. ANSI documents can be obtained from the American National Standards Institute, 1430 Broadway, New York, NY 10018, USA.

1.3.2 Application Conformance

All applications claiming conformance to this standard shall use only ANSI X3.9-1978 (FORTRAN 77) as described in 1.3.3.1 and shall fall within one of the following categories:

1.3.2.1 Strictly Conforming POSIX.9 Application

A Strictly Conforming POSIX.9 Application is an application that requires only the facilities in POSIX.9 and the facilities described in POSIX.1 {2} 1.3 and the language standard described in 1.3.3.1 for a *Strictly Conforming POSIX.1 Application*. Such an application shall accept any behavior described in this standard as *unspecified* or *implementation-defined*.

1.3.2.2 Conforming POSIX.9 Application

An IEEE Conforming POSIX.9 Application is an application that uses only the facilities described in this standard and the facilities described in POSIX.1 {2} 1.3 for a *Conforming POSIX.1 Application*.

1.3.3 Language-Dependent Services for the FORTRAN 77 Programming Language

ANSI X3.9-1978 {3} (FORTRAN 77), will provide the definition of any FORTRAN 77 language-dependent features used by POSIX.9. Section 8 provides new facilities and amplifications to facilities provided by the FORTRAN 77 standard. Any implementation claiming conformance to POSIX.9 shall provide the facilities of the FORTRAN 77 standard {3} that are referenced in Section 8 of POSIX.9 and any additions and amplifications required by Section 8 and 1.3.3.1.

Although POSIX.9 references FORTRAN 77 features to describe its own requirements, conformance to the FORTRAN 77 standard {3} is unnecessary for conformance to this standard. Any Fortran implementation that does not conflict with FORTRAN 77 and provides the facilities stipulated in Section 8 and 1.3.3.1 may claim conformance. However, it shall clearly state that its Fortran language does not conform to the FORTRAN 77 standard.

NOTE — FORTRAN 77 is considered to be contained within the ISO/IEC 1539: 1991 (Fortran 90), i.e., all features in FORTRAN 77 are considered to be part of Fortran 90. While Fortran 90 features need not be acceptable to FORTRAN 77, FORTRAN 77 features are acceptable under Fortran 90. To be able to use the Fortran 90 features, a separate bindings standard for Fortran 90 will be developed at a later time.

1.3.3.1 FORTRAN 77 Language Binding

ANSI X3.9-1978 {3} (FORTRAN 77) is used as the basis for this FORTRAN 77 language bindings to ISO/IEC 9945-1: 1990 {2}. Implementations claiming conformance to this standard must supply the FORTRAN 77 features required by this document, such as the intrinsic function facility.

One extension to FORTRAN 77 is required by POSIX.9. FORTRAN 77 specifies that “a symbolic name takes the form of one to six letters or digits, the first of which must be a letter.” This document assumes that the language implementation can accept specified symbolic names that are longer than six characters. Subroutine and function names, in particular, are assumed to be longer than six characters by this standard. Furthermore, to permit a FORTRAN 77 implementation to claim conformance to POSIX.1 {2}, names that differ in or before the 31st character position are required to be recognized as distinct names by the language implementation (see POSIX.1 {2} 1.3.5).

1.3.4 Other Language Related Specifications

The FORTRAN 77 standard {3} specifies that at least the 49 defined FORTRAN 77 characters shall exist with some FORTRAN 77-specified ordering requirements in an implementation-defined collating sequence. The FORTRAN 77 functions *CHAR()* and *ICHAR()* shall perform conversions based on that collating sequence.

A FORTRAN 77 character in a POSIX.9-conforming implementation shall be capable of representing all values of a byte define by POSIX.1 {2}. The POSIX.9-conforming implementation shall use a collating sequence which conforms to the FORTRAN 77 standard {3}.

2. Terminology and General Requirements

2.1 Conventions

2.1.1 Typographical Conventions

The following typographical conventions are used in this standard:

- 1) The *italic* font is used for:
 - Cross references to defined terms within 1.2, 2.2.1, and 2.2.2, or within these sections in POSIX.1 {2}.
 - Symbolic parameters in Synopsis subclauses and in the text that are generally substituted with real values by the application.
 - FORTRAN language data types, variable names, and subroutine/function names (except in Synopsis subclauses)
- 2) The **bold** font is used with a word in all capital letters, such as **PATH** to represent an environment variable, as described in 2.6. It is also used for the term “**NULL** pointer.”
- 3) The **constant-width** (Courier) font is used:
 - For FORTRAN 77 language data types and function names within function Synopsis subclauses
 - To illustrate examples of system input or output where exact usage is depicted
 - For references to C-language syntax and headers
- 4) HELVETICA font is used in 8.3 to represent a “generic” data type for which the appropriate actual FORTRAN 77 data type is substituted.
- 5) Symbolic constants returned by many functions and subroutines as error numbers are represented as: [ERRNO]
See 2.4.
- 6) Symbolic constants or limits defined in certain POSIX.1 {2} headers are represented as: {LIMIT}
See 2.8 and 2.9.

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

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

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

2.1.2 Namespace Conventions

The following naming conventions are used in this standard:

2.1.2.1 subroutine naming:

This standard defines a FORTRAN 77 subroutine interface to POSIX.1 {2} system calls and language-specific service routines. This standard prefixes the names of the POSIX.1 {2} system calls and service routines with the characters

PXF to create a unique name for the corresponding FORTRAN 77 procedures. For consistency, the names for the other service subroutines defined in this standard are also prefixed with the same characters.

2.1.2.2 function naming:

This standard defines a FORTRAN 77 function interface to the functionality provided by certain POSIX.1 {2} macros and service functions. All service functions in this standard that return an integer value are prefixed with the characters *IPXF*.

2.1.2.3 argument naming:

The names of all integer items in the actual argument list of each defined procedure statement in the Synopsis sections begin with one of the letters I, L, M, or N. The names of all items that are *structure handles* or *subroutine handles* (see 2.2.2) in the actual argument list are prefixed with the letter J.

2.2 Definitions

2.2.1 Terminology

For the purposes of this standard, the following definitions from POSIX.1 {2} apply:

2.2.1.1 conformance document: A document provided by an implementor that contains implementation details as described in POSIX.1 {2} 1.3.1.2.

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

2.2.1.3 may: An indication of an optional feature.

With respect to implementations, the word *may* is to be interpreted as an optional feature that is not required in this standard but can be provided. With respect to Strictly Conforming POSIX.9 Applications, the word *may* means that the optional feature shall not be used.

2.2.1.4 obsolescent: An indication that a certain feature may be considered for withdrawal in future revisions of this standard.

Obsolescent features are retained in this version because of their widespread use. Their use in new applications is discouraged.

2.2.1.5 shall: An indication of a requirement on the implementation or on Strictly Conforming POSIX.9 Applications, where appropriate.

2.2.1.6 should:

- 1) With respect to implementations, an indication of an implementation recommendation, but not a requirement.
- 2) With respect to applications, an indication of a recommended programming practice for applications and a requirement for Strictly Conforming POSIX.9 Applications.

2.2.1.7 supported: A condition regarding optional functionality.

Certain functionality in this standard is optional, but the interfaces to that functionality are always required. If the functionality is *supported*, the interfaces work as specified by this standard (except that they do not return the error condition indicated for the unsupported case). If the functionality is not *supported*, the interface shall always return the indication specified for this situation.

2.2.1.8 system documentation: All documentation provided with an implementation, except the conformance document.

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

2.2.1.9 undefined: An indication that this standard imposes no portability requirements on an application's use of an indeterminate value or its behavior with erroneous program constructs or erroneous data.

Implementations (or other standards) may specify the result of using that value or causing that behavior. An application using such behaviors is using extensions, as defined in POSIX.1 {2} 1.3.

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

Implementations (or other standards) may specify the result of using that value or causing that behavior. An application requiring a specific behavior, rather than tolerating any behavior when using that functionality, is using extensions, as defined in POSIX.1 {2} 1.3.

2.2.2 General Terms

In addition to those terms defined in ISO/IEC 9945-1: 1990 (see POSIX.1 {2} 2.2), the terms defined in this section are used in this standard.

2.2.2.1 component: A member, element, or field of a structure.

2.2.2.2 handle: An integer value that refers to a structure handle or subroutine handle.

Unless otherwise specified in this standard, *handle* refers to a structure handle.

2.2.2.3 structure handle: An integer value that refers to a unique instance of a structure.

An existing (structure) handle is a handle that references an existing instance of a structure.

2.2.2.4 subroutine handle: An integer value that refers to a unique instance of a subroutine.

2.2.2.5 intent: A description of whether the actual argument is used by a defined subprogram as an *input* argument (intent=IN), as an *output* argument (intent=OUT), or as both an input and output argument (intent=INOUT).

2.2.2.6 newline delimited: An indication that the newline character is used as a delimiter.

2.2.2.7 POSIX-based FORTRAN I/O file: A FORTRAN 77 file associated with a POSIX.1 {2} file descriptor that is connected to a FORTRAN 77 unit.

2.2.2.8 significant trailing blanks: One or more blanks at the end of a character string that are intended to be a meaningful part of the contents of the string.

Unlike strings in the C language, character variables in FORTRAN 77 are of a fixed length and are padded with blanks. That is, if a character variable is assigned a value that contains fewer characters than declared, the remainder of the variable is filled with blanks. Because of this characteristic, it is difficult to determine the difference between a string (e.g., a value assigned to an environment variable) that contains trailing blanks that are part of the string (significant trailing blanks) and a string for which the trailing blanks are only FORTRAN 77-required padding. For example, the strings "myprompt" and "mypromptΔ" (where Δ represents a significant blank that is part of the string) can be indistinguishable, but they are different legally valid prompts.

2.2.2.9 structure: An aggregate data type that allows the grouping of multiple data elements of possibly differing type into a single unit.

Two common implementations of structures are the *struct* in C and the *record* in Pascal. FORTRAN 77 provides no such aggregate data type.

2.2.2.10 text file: A file that contains characters organized into one or more lines.

The lines shall not contain NUL characters and none shall exceed {LINE_MAX} bytes in length, including the newline.

2.2.3 Abbreviations

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

2.2.3.1 POSIX.1: This standard assumes and uses POSIX.1 {2} to mean ISO/IEC 9945-1: 1990 {2}. References to sections and terms in that standard will be indicated using this term, e.g., “POSIX.1 {2} 2.3” will imply a reference to Section 2.3 of ISO/IEC 9945-1: 1990 and “POSIX.1 {2} *fork()*” will imply the function *fork()* described in that standard.

2.2.3.2 POSIX.9: This standard.

2.2.3.3 FORTRAN 77: This standard assumes and uses FORTRAN 77 to mean ANSI X3.9-1978 {3} plus the extension for long identifier names described in 1.3.3.1.

2.3 FORTRAN 77 Language Bindings Concepts

The following subsections present many of the design issues addressed in the development of this FORTRAN 77 Bindings standard. The discussion here is intended to be largely functional, i.e., describing only specific problems and their solutions. The accompanying rationale (Annex A) discusses in more detail the design objectives and alternatives that were considered in the development of this standard. With the exception of 2.3, the sections of the rationale correspond directly to the sections of this standard and to the sections of the POSIX.1 {2} rationale.

Because the POSIX.1 {2} system services are defined in the C language and use many language features that are not available in FORTRAN 77, many of the issues presented below are the result of differences and incompatibilities between these two languages. The main goals were to achieve access to all required POSIX.1 {2} functionality while following FORTRAN 77 as closely as possible and to allow consistent treatment of the exceptional cases.

2.3.1 System Headers

System headers containing definitions of *symbolic constants* and *macros* are used extensively throughout POSIX.1 {2}. These header files are intended for inclusion in application programs through the use of the C-language `#include` mechanism; however, FORTRAN 77 provides no similar inclusion capability, so methods had to be devised to allow the required header definitions to be accessed from FORTRAN 77 programs.

2.3.1.1 Symbolic Constants

The POSIX.1 {2} system headers contain the definitions of symbolic constants intended for use throughout the POSIX.1 {2} programming environment. These symbolic constants can be accessed from FORTRAN 77 with defined procedures. An overview of these procedures is given below:

- An integer function that returns the value of the named constant. This function can be used as an in-line call and provides no error checking.
- A logical function that indicates if the named constant is defined. This functionality is similar to the *feature test macro* in a C-based POSIX.1 {2} system.
- A subroutine that returns an argument containing the named constant. This subroutine provides full error checking.

Further information, including definitions of the interfaces that shall be provided, is given in 8.2.

2.3.1.2 Macros

Where functionality in POSIX.1 {2} is provided by macros, this standard specifies FORTRAN 77 functions. Definitions and descriptions for the functions that provide the functionality of those POSIX.1 {2} macros are included in the appropriate sections of this standard.

2.3.2 Data Types

Incompatibilities between the data types of the C and FORTRAN 77 languages caused many issues in the development of this standard. These incompatibilities can be divided into two categories: language-defined data types and data type definition capabilities.

2.3.2.1 Primitive Data Types

FORTRAN 77 does not provide a facility for type definition such as the *typedef* in the C language. Each of the primitive data types defined in POSIX.1 {2}, as well as any additional implementation-defined primitive types, is equated with a corresponding FORTRAN 77 intrinsic type in order to be used in FORTRAN 77 applications. Specifically, the FORTRAN 77 INTEGER data type shall be used as a substitute for defined arithmetic types (see POSIX.1 {2} 2.5).

2.3.2.2 Numeric Range of Integer Data

Many functions defined in POSIX.1 {2} make use of the *unsigned integer* data type provided by C to double the range of an argument or returned value. FORTRAN 77 does not provide such a data type. An implementation of this standard may choose to utilize an available sign bit of the FORTRAN 77 INTEGER data type to extend the range of such values to a range equivalent to that provided by the C bindings. Instances where this is allowable are indicated throughout the text of this standard. A support routine is defined (see 8.11) to allow these extended-range integer values to be compared.

2.3.2.3 Aggregate Data Types

Many of the service interfaces defined in POSIX.1 {2} require the use of aggregate data types that do not map to FORTRAN 77. FORTRAN 77 does not provide any mechanism for the construction or use of aggregate data types, creating a serious conflict.

The solution to this problem in the FORTRAN 77 bindings involves the use of *data abstraction*: Through the use of additional subroutines to access and manipulate the aggregate data, the underlying data structures are largely hidden from the FORTRAN 77 source code. It is the responsibility of the FORTRAN 77 programmer to maintain variables corresponding to the individual components of the aggregate data, but the programmer need not worry about the details of the actual implementation of the aggregate. The basic model of this data abstraction is used as follows:

- The programmer calls a subroutine to “create” an instance of the desired aggregate data type; this subroutine returns a handle that the programmer subsequently uses in order to reference and/or manipulate the data. The handle is guaranteed to fit in an integer variable, and a valid handle is guaranteed to be nonzero.
- The programmer uses additional subroutines to load values into or extract values from the aggregate data. These subroutines are passed the handle of the desired aggregate and the name of the specific component that is to be accessed. Notice that the programmer has direct control over only one component at a time.
- When an instance of an aggregate is no longer required, a subroutine can be called to release it.
- A subroutine is defined to duplicate contents of an instance of an aggregate.

Further information, including definitions of the subroutines that shall be provided, is given in 8.3.

2.3.2.3.1 List of Aggregate Data Types

The structure types shown in Table 2.1 shall be accessible through the techniques described previously in this section. The components of these structures are enumerated and described in the section indicated.

Table 2.1—Structure Types

Structure Name	Reference
sigset	3.3.1 and 3.3.3
sigaction	3.3.4
utsname	4.4.1
tms	4.5.2
dirent	5.1.1
stat	5.6.1
utimbuf	5.6.6
flock	6.5.2
termios	7.1.2
group	9.2.1
passwd	9.2.2

An implementation may provide additional structures to be used in an implementation-defined manner. Any such structures may be defined by an implementation to be able to be manipulated using these same structure manipulation subroutines.

POSIX.1 {2} allows the implementation to support additional components of many of these structures. In an implementation of POSIX.9 that corresponds to an implementation of POSIX.1 {2} that allows such extensions, access to these additional components shall be provided using these same structure manipulation subroutines.

2.3.2.4 Character Variables and String Manipulation

Data contained in a FORTRAN 77 *string* [a dummy argument declared as CHARACTER*(*)] is padded with blanks if necessary to match the declared string length. Because of this, it is difficult to differentiate between string data that is intended to contain trailing blanks and data that has simply been padded with blanks in order to match the declared string variable length. To allow this distinction, an extra argument is passed to or from procedures that have a string argument.

For procedures in which the string is *returned from* the system, this extra argument shall contain the *actual* length of the data assigned to the string. This length value can be zero, which indicates the equivalent of a **NULL** string indicating that the value of the string is undefined. If the length of the character argument is insufficient to contain the data to be returned from the system, *IERROR* shall be set to [ETRUNC], the data shall be truncated to fit the string, and the length argument shall contain the original length of the data before truncation.

For procedures in which the string is being *passed to* the system, this extra argument contains the *intended* length of the string contents, which is not necessarily the fixed, maximum length of the string variable. A value of zero passed as the length of the string data shall indicate that trailing blanks are to be stripped and ignored, or, if the string contains only blanks, shall indicate the equivalent of a **NULL** string.

2.3.2.5 Pointers

C-language pointers are used throughout POSIX.1 {2}; however, FORTRAN 77 does not have a pointer data type. In cases where POSIX.1 {2} specifies functionality dependent on the use or detection of a **NULL** pointer, the behavior has been modified slightly in this binding.

All uses of structures in POSIX.1 {2} are through pointers, i.e., structures are passed by reference to system functions. In conjunction with the methods defined in this binding for accessing and manipulating structured data, an object called a *handle* is used in the FORTRAN 77 interfaces where POSIX.1 {2} uses pointers to structures. A handle is an abstract reference to the aggregate data and does not require any direct manipulation by the FORTRAN 77 programmer. See 2.3.2.3 and 8.3 for further discussion of aggregate data and the use of handles.

Although FORTRAN 77 permits subroutines and functions declared as `EXTERNAL` to be passed as arguments to another procedure, there is no way in FORTRAN 77 to store a pointer to a subroutine for later use. This functionality shall be provided by the subroutines described in 8.4.

2.4 Error Numbers

Most functions in POSIX.1 {2} provide an error number in the external system variable *errno*, which is defined in the C language as:

```
extern int errno;
```

In this standard, the interface specification for subroutines that can result in error conditions contains an extra out argument, *IERROR*. Unless otherwise specified, a value of zero returned in this argument indicates that no error has occurred and a nonzero value indicates that an error *has* occurred. In this case, the value of other out arguments are undefined unless otherwise specified.

The following symbolic names identify additional errors that can occur in the use of this standard. The values represented by these names shall be unique and shall not conflict with error numbers specified in POSIX.1 {2}.

[ENONAME]	Invalid constant, structure, or component name.
[ENOHANDLE]	Handle not created.
[ETRUNC]	The declared length of the out character argument is insufficient to contain the string to be returned. (See 2.3.2.4.)
[EARRAYLEN]	For <i>get</i> routines, the number of array elements to be returned exceeds <i>IALEN</i> , and only the first <i>IALEN</i> elements of the array argument have been set. For <i>set</i> routines, <i>IALEN</i> exceeds the number of array elements in the target array. Only the available elements of the array in the target array have been set.
[EEND]	End of file, record, or directory stream has been encountered.

2.5 Primitive System Data Types

Because all of the primitive system data types shall be *arithmetic* types (see POSIX.1 {2} 2.5), the FORTRAN 77 INTEGER data type shall be used as a substitute for each of the listed types. However, when a primitive data type is defined in the C bindings to POSIX.1 {2} as an unsigned integer, other issues may arise. See 2.3.2.1 and 2.3.2.2 for further discussion.

2.6 Environment Description

The individual members of the environment (see POSIX.1 {2} 2.6) are examined using the *PXFGETENV()* subroutine, modified using the *PXFSETENV()* subroutine, and cleared by the *PXFCLEARENV()* subroutine (see 4.6.1).

2.7 FORTRAN 77 Language Definitions

FORTRAN language terms and symbols used in this standard are defined by FORTRAN 77.

2.8 Numerical Limits

2.8.1 FORTRAN 77 Language Limits

Certain limits used in this standard are defined in the FORTRAN 77 programming language.

2.8.2 Minimum Values

The symbolic constants specifying minimum values described in POSIX.1 {2} 2.8.2 shall be accessible through calls to any of the *PXFCONST* procedures (see 8.2).

2.8.3 Run-Time Increasable Values

The magnitude limitations specified in POSIX.1 {2} 2.8.3 shall be accessible through a call to *PXFSYSCONF*() (see 4.8).

2.8.4 Run-Time Invariant Values (Possible Indeterminate)

The run time invariant values specified in POSIX.1 {2} 2.8.4 shall be accessible through a call to *PXFSYSCONF*() (see 4.8).

2.8.5 Pathname Variable Values

The pathname variable values specified in POSIX.1 {2} 2.8.5 shall be accessible through a call to *PXFPATHCONF*() (see 5.7).

2.8.6 Invariant Values

The invariant values specified in POSIX.1 {2} 2.8.6 shall be accessible through a call to *PXFSYSCONF*() (see 4.8).

2.9 Symbolic Constants

The symbolic constants defined in POSIX.1 {2} (see POSIX.1 {2} 2.9) and POSIX.9 shall be accessible through calls to any of the *PXFCONST*() procedures (see 8.2).

2.9.1 Constants for FORTRAN 77 I/O to STDIO Translation

The following symbolic constants shall be accessible through calls to any of the *PXFCONST*() procedures (see 8.2).

STDIN_UNIT	The value of the FORTRAN 77 unit identifier associated with a preconnected input file.
STDOUT_UNIT	The value of the FORTRAN 77 unit identifier associated with a preconnected output file.
STDERR_UNIT	The value of the FORTRAN 77 unit identifier associated with a preconnected error file.

The values of these constants shall be integers in the range 0–9. Portable applications using units for other files should use values outside these ranges.

3. Process Primitives

3.1 Process Creation and Execution

3.1.1 Process Creation

Subroutine: *PXFFORK()*

3.1.1.1 Synopsis

```
SUBROUTINE PXFFORK ( IPID, IERROR)
  INTEGER IPID, IERROR
```

3.1.1.2 Description

The *PXFFORK()* subroutine shall provide the same functionality as the POSIX.1 {2} function *fork()* (see POSIX.1 {2} 3.1) except that files opened with the FORTRAN 77 OPEN statement are not required to have file descriptors (see 8.5).

Arguments for *PXFFORK()* correspond to the arguments for *fork()*, as shown in Table 3.1.

Table 3.1—Arguments for *PXFFORK()*

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
IPID	ret_value	OUT	
IERROR	ret_value/errno	OUT	

3.1.1.3 Errors

Possible error conditions for *PXFFORK()* are identical to those for the POSIX.1 {2} function *fork()*. Under the circumstances specified by POSIX.1 {2}, the argument *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

3.1.2 Execute a File

Subroutines: *PXFEXECV()*, *PXFEXECVE()*, *PXFEXECVP()*

3.1.2.1 Synopsis

```
SUBROUTINE PXFEXECV (PATH, LENPATH, ARGV, LENARGV, IARGC, IERROR)
  INTEGER LENPATH, LENARGV(0:IARGC-1), IARGC, IERROR
  CHARACTER*(*) PATH, ARGV(0:IARGC-1)
```

```

SUBROUTINE PXFEXECVE (PATH, LENPATH, ARGV, LENARGV, IARGC,
+  ENV, LENENV, IENVC, IERROR)
  INTEGER LENPATH, LENARGV(0:IARGC-1), IARGC, LENENV(IENVC), IENVC, IERROR
  CHARACTER*(*) PATH, ARGV(0:IARGC-1), ENV(IENVC)
```

```

SUBROUTINE PXFEXECVP (FILE, LENFILE, ARGV, LENARGV, IARGC, IERROR)
INTEGER LENFILE, LENARGV(0:IARGC-1), IARGC, IERROR
CHARACTER*(*) FILE, ARGV(0:IARGC-1)

```

3.1.2.2 Description

The *PXFEXECV* subroutines shall provide the same functionality as the corresponding POSIX.1 {2} *exec* functions in POSIX.1 {2} (see POSIX.1 {2} 3.1).

The lengths of the *ARGV* and *ENV* arrays are explicitly passed in the arguments *IARGC* and *IENV* respectively. The arrays *ARGV* and *LENARGV* shall be dimensioned at least as large as *IARGC*, and the arrays *ENV* and *LENENV* shall be dimensioned at least as large as *IENV*. While these arrays may be dimensioned greater than required, the arguments *IARGC* and *IENV* specify the number of the elements at the beginning of the respective arrays that are to be used by the subroutine. The string length of each element of the *ARGV* and *ENV* arrays is passed in the corresponding element of the *LENARGV* and *LENENV* arrays.

Arguments for these subroutines correspond to the arguments for the corresponding POSIX.1 {2} process creation and execution functions, as shown in Table 3.2.

Table 3.2—Arguments for the *PXFEXEC...()* Subroutines

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
PATH	path	IN	
LENPATH	--	IN	Length of PATH; see 2.3.2.4
FILE	file	IN	
LENFILE	--	IN	Length of FILE; see 2.3.2.4
ARGV	argv	IN	
IARGC	--	IN	Number of elements in ARGV
LENARGV	--	IN	Length of elements in ARGV; see 2.3.2.4
ENV	envp	IN	
IENV	--	IN	Number of elements in ENV
LENENV	--	IN	Length of elements in ENV; see 2.3.2.4
IERROR	ret_value/errno	OUT	

3.1.2.3 Errors

Possible error conditions for the *PXFEXECV()* family of subroutines are identical to those for the POSIX.1 {2} *exec()* family. Under the circumstances specified by POSIX.1 {2}, the argument *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

3.2 Process Termination

Process termination shall provide the functionality defined by the POSIX.1 {2} function *_exit()* (see POSIX.1 {2} 3.2.2). There are two kinds of process termination, normal and abnormal. Normal termination occurs by the execution of the FORTRAN 77 END statement in the FORTRAN 77 main program, the FORTRAN 77 STOP statement, or the

PXFFASTEXIT() or *PXFEXIT()* subroutine (see 3.2.2 and 8.12). Abnormal termination occurs when certain signals are received, as defined in POSIX.1 {2} 3.3.

A parent process may suspend its execution to wait for the termination of a child process with the *PXFWAIT()* or *PXFWAITPID()* subroutines.

3.2.1 Wait for Process Termination

Subroutines: *PXFWAIT()*, *PXFWAITPID()*

3.2.1.1 Synopsis

```
SUBROUTINE PXFWAIT ( ISTAT, IRETPID, IERROR)
  INTEGER ISTAT, IRETPID, IERROR
```

```
SUBROUTINE PXFWAITPID ( IPID, ISTAT, IOPTIONS, IRETPID, IERROR)
  INTEGER IPID, ISTAT, IOPTIONS, IRETPID, IERROR
```

3.2.1.2 Description

The *PXFWAIT()* and *PXFWAITPID()* subroutines shall provide the same functionality as the POSIX.1 {2} functions *wait()* and *waitpid()* (see POSIX.1 {2} 3.2).

The value for the *IOPTIONS* arguments to the *PXFWAITPID()* subroutine is based on the symbolic constants defined for *waitpid()*. These constants shall be accessible through any of the *PXFCNST()* procedures (see 8.2). These values of the symbolic constants shall be distinct and can be combined with the use of the inclusive OR function (see 8.7). Arguments for *PXFWAIT()* and *PXFWAITPID()* correspond to the arguments for *wait()* and *waitpid()*, as shown in Table 3.3.

Table 3.3—Arguments for *PXFWAIT()* and *PXFWAITPID()*

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
ISTAT	stat_loc	OUT	
IPID	pid	IN	
IRETPID	ret_value	OUT	
IOPTIONS	options	IN	
IERROR	ret_value/errno	OUT	

The following functions may be used to interpret the *ISTAT* argument, as defined in POSIX.1 {2} 3.2.

```
LOGICAL FUNCTION PXFWIFEXITED ( ISTAT)
  INTEGER ISTAT
```

```
INTEGER FUNCTION IPXFWEXITSTATUS ( ISTAT)
  INTEGER ISTAT
```

```
LOGICAL FUNCTION PXFWIFSIGNALED ( ISTAT)
  INTEGER ISTAT
```

```
INTEGER FUNCTION IPXFWTERMSIG ( ISTAT )
INTEGER ISTAT
```

```
LOGICAL FUNCTION PXFWIFSTOPPED ( ISTAT )
INTEGER ISTAT
```

```
INTEGER FUNCTION IPXFWSTOPSIG ( ISTAT )
INTEGER ISTAT
```

3.2.1.3 Errors

Possible error conditions for *PXFWAIT()* and *PXFWAITPID()* are identical to those for the POSIX.1 {2} functions *wait()* and *waitpid()*. *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

3.2.2 Terminate a Process

Subroutine: *PXFFASTEXIT()*

3.2.2.1 Synopsis

```
SUBROUTINE PXFFASTEXIT ( ISTATUS )
INTEGER ISTATUS
```

3.2.2.2 Description

The *PXFFASTEXIT()* subroutine shall provide the same functionality as the POSIX.1 {2} function *_exit()* (see POSIX.1 {2} 3.2). There is no possible return from *PXFFASTEXIT()*, and no *IERROR* argument is defined for *PXFFASTEXIT()*. Arguments for *PXFFASTEXIT()* correspond to the arguments for *_exit()*, as shown in Table 3.4.

Table 3.4—Arguments for *PXFFASTEXIT()*

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
ISTATUS	status	IN	

3.3 Signals

3.3.1 Signal Concepts

3.3.1.1 Signal Names

The values for use with the signal procedures are based on the symbolic constants defined for POSIX.1 {2} signals. These constants shall be accessible through any of the *PXFCONST()* procedures (see 8.2).

The symbolic constants *SIG_DFL* and *SIG_IGN* represent values that shall not be identical to any value returned by *PXFGETSUBHANDLE()* (see 8.4 and 3.3.1.3). When used as the handle for the signal-catching subroutine, they shall cause the signal-specific default action or ignore signal action respectively.

The subroutine *PXFSTRUCTCREATE()* with the string ‘sigset’ given as the *STRUCTNAME* argument may be used to obtain an instance of the *sigset_t* type as defined in POSIX.1 {2} 3.3.1. There are no defined components of this

structure, and the contents of the structure may not be altered with the structure-component manipulation subroutines. Instead, the subroutines defined in 3.3.3 shall be used.

3.3.1.2 Signal Generation and Delivery

3.3.1.3 Signal Actions

On delivery of a signal (see POSIX.1 {2} 3.3), the system may call a user-defined signal-catching subroutine (see 3.3.4). This signal-catching subroutine shall be defined with a single integer argument. The argument contains the number of the signal being delivered.

3.3.1.4 Signal Effects on Other Subroutines

Signals may affect the behavior of certain procedures defined by this standard if delivered to a process while it is executing such a procedure. Specifically, nonzero values of *IERROR* for each of the system services are not guaranteed to be reliable in the presence of signals.

3.3.2 Send a Signal to a Process

Subroutine: *PXFKILL()*

3.3.2.1 Synopsis

```
SUBROUTINE PXFKILL ( IPID, ISIG, IERROR )
  INTEGER IPID, ISIG, IERROR
```

3.3.2.2 Description

The *PXFKILL()* subroutine shall provide the same functionality as the POSIX.1 {2} function *kill()* (see POSIX.1 {2} 3.3).

The value of the desired signal (specified by *ISIG*) shall be accessible through calls to any of the *PXFCONST()* procedures (see 8.2). Arguments for *PXFKILL()* correspond to the arguments for *kill()*, as shown in Table 3.5.

Table 3.5—Arguments for *PXFKILL()*

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
IPID	pid	IN	
ISIG	sig	IN	
IERROR	ret_value/errno	OUT	

3.3.2.3 Errors

Possible error conditions for *PXFKILL* are identical to those for the POSIX.1 {2} function *kill()*. *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

3.3.3 Manipulate Signal Sets

Subroutines: *PXFSIGEMPTYSET()*, *PXFSIGFILLSET()* *PXFSIGADDSET()*, *PXFSIGDELSET()*, *PXFSIGISMEMBER()*

3.3.3.1 Synopsis

```

SUBROUTINE PXFSIGEMPTYSET (JSIGSET, IERROR)
INTEGER JSIGSET, IERROR

SUBROUTINE PXFSIGFILLSET (JSIGSET, IERROR)
INTEGER JSIGSET, IERROR

SUBROUTINE PXFSIGADDSET (JSIGSET, ISIGNO, IERROR)
INTEGER JSIGSET, ISIGNO, IERROR

SUBROUTINE PXFSIGDELSET (JSIGSET, ISIGNO, IERROR)
INTEGER JSIGSET, ISIGNO, IERROR

SUBROUTINE PXFSIGISMEMBER (JSIGSET, ISIGNO, ISMEMBER, IERROR)
INTEGER JSIGSET, ISIGNO, IERROR
LOGICAL ISMEMBER

```

3.3.3.2 Description

These subroutines shall provide the same functionality as the equivalent POSIX.1 {2} signal set manipulation functions (see POSIX.1 {2} 3.3).

The *PXFSIGISMEMBER()* procedure shall return a logical value *.TRUE.* in the argument *ISMEMBER* if the specified signal is a member of the specified set or a value of *.FALSE.* if it is not.

Applications shall call either *PXFSIGEMPTYSET()* or *PXFSIGFILLSET()* at least once for each *sigset* structure prior to any other use of that structure. If the structure is not initialized in this way, the results are undefined.

This standard defines *sigset* as a structure. An instance of a *sigset* shall be created using *PXFSTRUCTCREATE()* before manipulation using these subroutines.

Arguments for these subroutines correspond to the arguments for the corresponding POSIX.1 {2} signal set manipulation functions, as shown in Table 3.6.

Table 3.6—Arguments for the *PXFSIG...()* Subroutines

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
<i>JSIGSET</i>	<i>set</i>	IN	1.
<i>ISMEMBER</i>	<i>ret_value</i>	OUT	
<i>ISIGNO</i>	<i>signo</i>	IN	
<i>IERROR</i>	<i>ret_value/errno</i>	OUT	
1. Handle obtained from <i>PXFSTRUCTCREATE</i> ('sigset',...); see 8.3.1.			

3.3.3.3 Errors

Possible error conditions for these subroutines are identical to those for the corresponding signal set manipulation functions defined in POSIX.1 {2}. *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

3.3.4 Examine and Change Signal Action

Subroutine: *PXFSIGACTION()*

3.3.4.1 Synopsis

```
SUBROUTINE PXFSIGACTION (ISIG, JSIGACT, JOSIGACT, IERROR)
  INTEGER ISIG, JSIGACT, JOSIGACT, IERROR
```

3.3.4.2 Description

The *PXFSIGACTION()* subroutine shall provide the same functionality as the POSIX.1 {2} function *sigaction()* (see POSIX.1 {2} 3.3). Arguments for *PXFSIGACTION()* correspond to the arguments for *sigaction()*, as shown in Table 3.7.

Table 3.7—Arguments for *PXFSIGACTION()*

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
ISIG	sig	IN	
JSIGACT	act	IN	1.
JOSIGACT	oact	OUT	1.
IERROR	ret_value/errno	OUT	

1. Handle obtained from *PXFSTRUCTCREATE* ('sigaction',...); see 8.3.1.

The functionality obtained in the POSIX.1 {2} function *sigaction()* by passing a **NULL** can be obtained in *PXFSIGACTION* by passing a handle argument with a value of zero.

The values of the symbolic constants *SIG_DFL* and *SIG_IGN* shall be accessible through calls to any of the *PXFCONST()* procedures (see 8.2) and can be used as values for the signal handler component. They shall cause the signal-specific default action or ignore signal action respectively, as defined by POSIX.1 {2} 3.3.1.3.

The *PXFSTRUCTCREATE()* subroutine (see 8.3.1) with the string 'sigaction' given as the *STRUCTNAME* argument shall be used to obtain a handle for an instance of the *sigaction* structure as defined in POSIX.1 {2} 3.3. Each component access shall require one of the following structure-component manipulation subroutines (see 8.3.2):

```
SUBROUTINE PXFINTGET(JSIGACTION, COMPNAM, IVALUE, IERROR)
  INTEGER JSIGACTION, IVALUE, IERROR
```

```
SUBROUTINE PXFINTSET(JSIGACTION, COMPNAM, IVALUE, IERROR)
  INTEGER JSIGACTION, IVALUE, IERROR
```

where *JSIGACTION* is a handle and *COMPNAM* is a character expression which evaluates to one of the component names shown in Table 3.8.

Table 3.8—Components for *sigaction* Structure

POSIX.1 Component	COMPNAM	Structure Procedure Used to Access
sa_handler	'sa_handler'	PXFINTGET,PXFINTSET
sa_mask	'sa_mask'	PXFINTGET,PXFINTSET
sa_flags	'sa_flags'	PXFINTGET,PXFINTSET

The *sa_handler* component shall be a subroutine handle obtained from a call to *PXFGETSUBHANDLE()* (see 8.4), obtained from a previous call to *PXFSIGACTION()*, or that shall contain the value of one of the symbolic constants *SIG_DFL* or *SIG_IGN*. The *sa_mask* component shall be a sigset structure handle (see 3.3) obtained from a call to *PXFSTRUCTCREATE()*.

Values of the *sa_flags* component can be used to modify the behavior of the signal specified in a call to *PXFSIGACTION()*. Values of *sa_flags* are composed of the flag bits used by the *sigaction()* function as defined in POSIX.1 {2} 3.3. The values of these flags shall be bitwise distinct and can be combined with the use of the inclusive OR function (see 8.7). The flag names are constants for which the values shall be accessible through calls to any of the *PXFCONST()* procedures (see 8.2).

3.3.4.3 Errors

Possible error conditions for *PXFSIGACTION()* are identical to those for the POSIX.1 {2} function *sigaction()*. Under the circumstances specified by POSIX.1 {2}, the argument *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

3.3.5 Examine and Change Blocked Signals

Subroutine: *PXFSIGPROCMASK()*

3.3.5.1 Synopsis

```
SUBROUTINE PXFSIGPROCMASK (IHOW, JSIGSET, JOSIGSET, IERROR)
  INTEGER IHOW, JSIGSET, JOSIGSET, IERROR
```

3.3.5.2 Description

The *PXFSIGPROCMASK()* subroutine shall provide the same functionality as the POSIX.1 {2} function *sigprocmask()* (see POSIX.1 {2} 3.3). Arguments for *PXFSIGPROCMASK()* correspond to the arguments for *sigprocmask()*, as shown in Table 3.9.

Table 3.9—Arguments for *PXFSIGPROCMASK()*

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
IHOW	how	IN	
JSIGSET	set	IN	1.
JOSIGSET	oset	OUT	1.
IERROR	ret_value/errno	OUT	
1. Handle obtained from PXFSTRUCTCREATE ('sigset',...); see 8.3.1.			

The functionality obtained in the POSIX.1 {2} function *sigprocmask()* by passing a **NULL** may be obtained in *PXFSIGPROCMASK* by passing a handle argument with a value of zero.

3.3.5.3 Errors

Possible error conditions for *PXFSIGPROCMASK()* are identical to those for the POSIX.1 {2} function *sigprocmask()*. *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

3.3.6 Examine Pending Signals

Subroutine: *PXFSIGPENDING()*

3.3.6.1 Synopsis

```
SUBROUTINE PXFSIGPENDING (JSIGSET, IERROR
  INTEGER JSIGSET, IERROR
```

3.3.6.2 Description

The *PXFSIGPENDING()* subroutine shall provide the same functionality as the POSIX.1 {2} function *sigpending()* (see POSIX.1 {2} 3.3). Arguments for *PXFSIGPENDING()* correspond to the arguments for *sigpending()*, as shown in Table 3.10.

Table 3.10—Arguments for *PXFSIGPENDING()*

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
JSIGSET	set	IN	1.
IERROR	ret_value/errno	OUT	
1. Handle obtained from PXFSTRUCTCREATE ('sigset',...); see 8.3.1.			

3.3.6.3 Errors

Possible error conditions for *PXFSIGPENDING()* are identical to those for the POSIX.1 {2} function *sigpending()*. *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

3.3.7 Wait for a Signal

Subroutine: *PXFSIGSUSPEND()*

3.3.7.1 Synopsis

```
SUBROUTINE PXFSIGSUSPEND (JSIGSET, IERROR)
  INTEGER JSIGSET, IERROR
```

3.3.7.2 Description

The *PXFSIGSUSPEND()* subroutine shall provide the same functionality as the POSIX.1 {2} function *sigsuspend()* (see POSIX.1 {2} 3.3). Arguments for *PXFSIGSUSPEND()* correspond to the arguments for *sigsuspend()*, as shown in Table 3.11.

Table 3.11—Arguments for *PXFSIGSUSPEND()*

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
JSIGSET	sigmask	IN	1.
IERROR	ret_value/errno	OUT	
1. Handle obtained from PXFSTRUCTCREATE ('sigset',...); see 8.3.1.			

3.3.7.3 Errors

Since the *PXFSIGSUSPEND()* subroutine suspends process execution indefinitely, there is no successful completion return value.

Possible error conditions for *PXFSIGSUSPEND()* are identical to those for the POSIX.1 {2} function *sigsuspend()*. If any of these conditions occur, the argument *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

3.4 Timer Operations

3.4.1 Schedule Alarm

Subroutine: *PXFALARM()*

3.4.1.1 Synopsis

```
SUBROUTINE PXFALARM (ISECONDS, ISECLEFT, IERROR)
  INTEGER ISECONDS, ISECLEFT, IERROR
```

3.4.1.2 Description

The *PXFALARM()* subroutine shall provide the same functionality as the POSIX.1 {2} function *alarm()* (see POSIX.1 {2} 3.4).

If there is a previous *PXFALARM()* request with time remaining, the number of seconds until the previous request would have generated a SIGALARM signal is returned in *ISECLEFT*. Otherwise, *ISECLEFT* shall contain a value of zero upon return.

Arguments for *PXFALARM()* correspond to the arguments for *alarm()*, as shown in Table 3.12.

Table 3.12—Arguments for *PXFALARM()*

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
ISECONDS	seconds	IN	
ISECLEFT	ret_value	OUT	1.
IERROR	ret_value/errno	OUT	
1. Value may exceed the range of a signed integer; see 2.3.2.2.			

3.4.1.3 Errors

POSIX.1 {2} does not specify any error conditions that are required to be detected for the *alarm()* function. Upon successful completion of *PXFALARM()*, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

3.4.2 Suspend Process Execution

Subroutine: *PXFPAUSE()*

3.4.2.1 Synopsis

```
SUBROUTINE PXFPAUSE (IERROR)
  INTEGER IERROR
```

3.4.2.2 Description

The *PXFPAUSE()* subroutine shall provide the same functionality as the POSIX.1 {2} function *pause()* (see POSIX.1 {2} 3.4). Arguments for *PXFPAUSE()* correspond to the arguments for *pause()*, as shown in Table 3.13.

Table 3.13—Arguments for *PXFPAUSE()*

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
IERROR	ret_value/errno	OUT	

3.4.2.3 Errors

Since the *PXFPAUSE()* subroutine suspends process execution indefinitely, there is no successful completion return value.

Possible error conditions for *PXFPAUSE()* are identical to those for the POSIX.1 {2} function *pause()*. If any of these conditions occur, the argument *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

3.4.3 Delay Process Execution

Subroutine: *PXFSLEEP()*

3.4.3.1 Synopsis

```
SUBROUTINE PXFSLEEP (ISECONDS, ISECLEFT, IERROR)
  INTEGER ISECONDS, ISECLEFT, IERROR
```

3.4.3.2 Description

The *PXFSLEEP()* subroutine shall provide the same functionality as the POSIX.1 {2} function *sleep()* (see POSIX.1 {2} 3.4).

If *PXFSLEEP()* returns because the requested time has elapsed, the value of *ISECLEFT* is set to zero. If *PXFSLEEP()* returns due to delivery of a signal, *ISECLEFT* shall contain upon return the unslept amount (the requested time minus the time actually slept) in seconds.

Arguments for *PXFSLEEP()* correspond to the arguments for *sleep()*, as shown in Table 3.14.

Table 3.14—Arguments for *PXFSLEEP()*

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
ISECONDS	seconds	IN	
ISECLEFT	ret_value	OUT	1.
IERROR	ret_value/errno	OUT	
1. Value may exceed the range of a signed integer; see 2.3.2.2.			

3.4.3.3 Errors

POSIX.1 {2} does not specify any error conditions that are required to be detected for the *sleep()* function. Upon successful completion of *PXFSLEEP()*, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4. Process Environment

4.1 Process Identification

4.1.1 Get Process and Parent Process IDs

Subroutines: *PXFGETPID()*, *PXFGETPPID()*

4.1.1.1 Synopsis

```
SUBROUTINE PXFGETPID (IPID, IERROR)
  INTEGER IPID, IERROR
```

```
SUBROUTINE PXFGETPPID (IPID, IERROR)
  INTEGER IPID, IERROR
```

4.1.1.2 Description

The *PXFGETPID()* and *PXFGETPPID()* subroutines shall provide the same functionality as the POSIX.1 {2} functions *getpid()* and *getppid()* (see POSIX.1 {2} 4.1). Arguments for *PXFGETPID()* and *PXFGETPPID()* correspond to the arguments for *getpid()* and *getppid()*, as shown in Table 4.1.

Table 4.1—Arguments for *PXFGETPID()* and *PXFGETPPID()*

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
IPID	ret_value	OUT	
IERROR	ret_value/errno	OUT	

4.1.1.3 Errors

POSIX.1 {2} does not specify any error conditions that are required to be detected for the *getpid()* function or the *getppid()* function. Upon successful completion of *PXFGETPID()* or *PXFGETPPID()*, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4.2 User Identification

4.2.1 Get Real User, Effective User, Real Group, and Effective Group IDs

Subroutines: *PXFGETUID()*, *PXFGETEUID()*, *PXFGETGID()*, *PXFGETEGID()*

4.2.1.1 Synopsis

```
SUBROUTINE PXFGETUID (IUID, IERROR)
  INTEGER IUID, IERROR
```

```
SUBROUTINE PXFGETEUID (IEUID, IERROR)
  INTEGER IEUID, IERROR
```

```
SUBROUTINE PXFGETGID (IGID, IERROR)
  INTEGER IGID, IERROR
```

```
SUBROUTINE PXFGETEGID (IEGID, IERROR)
  INTEGER IEGID, IERROR
```

4.2.1.2 Description

The *PXFGETUID()*, *PXFGETEUID()*, *PXFGETGID()*, and *PXFGETEGID()* subroutines shall provide the same functionality as the POSIX.1 {2} functions *getuid()*, *geteuid()*, *getgid()*, and *getegid()* (see POSIX.1 {2} 4.2). Arguments for these subroutines correspond to the arguments for the corresponding POSIX.1 {2} user identification functions, as shown in Table 4.2.

Table 4.2—Arguments for the *PXFGET...ID()* Subroutines

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
IUID	ret_value	OUT	
IEUID	ret_value	OUT	
IGID	ret_value	OUT	
IEGID	ret_value	OUT	
IERROR	ret_value/errno	OUT	

4.2.1.3 Errors

POSIX.1 {2} does not specify any error conditions that are required to be detected for the *get...id()* family of functions. Upon successful completion of any of the *PXFGET...ID()* family of subroutines, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4.2.2 Set User and Group IDs

Subroutines: *PXFSETUID()*, *PXFSETGID()*

4.2.2.1 Synopsis

```
SUBROUTINE PXFSETUID (IUID, IERROR)
  INTEGER IUID, IERROR
```

```
SUBROUTINE PXFSETGID (IGID, IERROR)
  INTEGER IGID, IERROR
```

4.2.2.2 Description

The *PXFSETUID()* and *PXFSETGID()* subroutines shall provide the same functionality as the POSIX.1 {2} functions *setuid()* and *setgid()* (see POSIX.1 {2} 4.2). Arguments for *PXFSETUID()* and *PXFSETGID()* correspond to the arguments for *setuid()* and *setgid()*, as shown in Table 4.3.

Table 4.3—Arguments for *PXFSETUID()* and *PXFSETGID()*

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
IUID	uid	IN	
IGID	gid	IN	
IERROR	ret_value/errno	OUT	

4.2.2.3 Errors

Possible error conditions for *PXFSETUID()* and *PXFSETGID()* are identical to those for the POSIX.1 {2} functions *setuid()* and *setgid()*. *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4.2.3 Get Supplementary Group IDs

Subroutine: *PXFGETGROUPS()*

4.2.3.1 Synopsis

```
SUBROUTINE PXFGETGROUPS (IGIDSETSIZE, IGROUPLIST, NGROUPS, IERROR)
  INTEGER IGIDSETSIZE, IGROUPLIST(IGIDSETSIZE), NGROUPS, IERROR
```

4.2.3.2 Description

The *PXFGETGROUPS()* subroutine shall provide the same functionality as the POSIX.1 {2} function *getgroups()* (see POSIX.1 {2} 4.2), including the special case behavior when the *IGIDSETSIZE* argument is zero.

Arguments for *PXFGETGROUPS()* correspond to the arguments for *getgroups()*, as shown in Table 4.4.

Table 4.4—Arguments for *PXFGETGROUPS()*

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
IGROUPLIST	grouplist	OUT	
IGIDSETSIZE	gidsetsize	IN	
NGROUPS	ret_value	OUT	
IERROR	ret_value/errno	OUT	

4.2.3.3 Errors

Possible error conditions for *PXFGETGROUPS()* are identical to those for the POSIX.1 {2} function *getgroups()*. *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4.2.4 Get User Name

Subroutine: *PXFGETLOGIN()*

4.2.4.1 Synopsis

```
SUBROUTINE PXFGETLOGIN (S, ILEN, IERROR)
  CHARACTER*(*) S
  INTEGER ILEN, IERROR
```

4.2.4.2 Description

PXFGETLOGIN() shall provide the same functionality as the POSIX.1 {2} function *getlogin()* (see POSIX.1 {2} 4.2). Arguments for *PXFGETLOGIN()* correspond to the arguments for *getlogin()*, as shown in Table 4.5.

Table 4.5—Arguments for *PXFGETLOGIN()*

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
S	ret_value	OUT	
ILEN	—	OUT	Length of S; see 2.3.2.4
IERROR	ret_value/errno	OUT	

4.2.4.3 Errors

POSIX.1 {2} does not specify any error conditions that are required to be detected for the *getlogin()* function. Upon successful completion of *PXFGETLOGIN()*, the argument *IERROR* shall be set to zero. If any of the following conditions occur, *PXFGETLOGIN()* shall set the argument to the corresponding value. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

[ETRUNC] The declared length of the argument *S* is insufficient to contain the string to be returned.
(See 2.3.2.4.)

4.3 Process Groups

4.3.1 Get Process Group ID

Subroutine: *PXFGETPGRP()*

4.3.1.1 Synopsis

```
SUBROUTINE PXFGETPGRP ( IPGRP, IERROR)
  INTEGER, IPGRP, IERROR
```

4.3.1.2 Description

The *PXFGETPGRP()* subroutine shall provide the same functionality as the POSIX.1 {2} function *getpgrp()* (see POSIX.1 {2} 4.3). Arguments to *PXFGETPGRP()* correspond to the arguments for *getpgrp()*, as shown in Table 4.6.

Table 4.6—Arguments for *PXFGETPGRP()*

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
IPGRP	ret_value	OUT	
IERROR	ret_value/errno	OUT	

4.3.1.3 Errors

POSIX.1 {2} does not specify any error conditions that are required to be detected for the *getpgrp()* function. Upon successful completion of *PXFGETPGRP()*, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4.3.2 Create Session and Set Process Group ID

Subroutine: *PXFSETSID()*

4.3.2.1 Synopsis

```
SUBROUTINE PXFSETSID (ISID, IERROR)
  INTEGER ISID, IERROR
```

4.3.2.2 Description

The *PXFSETSID()* subroutine shall provide the same functionality as the POSIX.1 {2} function *setsid()* (see POSIX.1 {2} 4.3). Arguments for *PXFSETSID()* correspond to the arguments for *setsid()*, as shown in Table 4.7.

Table 4.7—Arguments for *PXFSETSID()*

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
ISID	ret_value	OUT	
IERROR	ret_value/errno	OUT	

4.3.2.3 Errors

Possible error conditions for *PXFSETSID()* are identical to those for the POSIX.1 {2} function *setsid()*. *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4.3.3 Set Process Group ID for Job Control

Subroutine: *PXFSETPGID()*

4.3.3.1 Synopsis

```
SUBROUTINE PXFSETPGID (IPID, IPGID, IERROR)
  INTEGER IPID, IPGID, IERROR
```

4.3.3.2 Description

The *PXFSETPGID()* subroutine shall provide the same functionality as the POSIX.1 {2} function *setpgid()* (see POSIX.1 {2} 4.3). Arguments for *PXFSETPGID()* correspond to the arguments for *setpgid()*, as shown in Table 4.8.

Table 4.8—Arguments for *PXFSETPGID()*

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
IPID	pid	IN	
IPGID	pgid	IN	
IERROR	ret_value/errno	OUT	

4.3.3.3 Errors

Possible error conditions for *PXFSETPGID()* are identical to those for the POSIX.1 {2} function *setpgid()*. *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4.4 System Identification

4.4.1 Get System Name

Subroutine: *PXFUNAME()*

4.4.1.1 Synopsis

```
SUBROUTINE PXFUNAME ( JUTSNAME, IERROR)
  INTEGER JUTSNAME, IERROR
```

4.4.1.2 Description

The *PXFUNAME()* subroutine shall provide the same functionality as the POSIX.1 {2} function *uname()* (see POSIX.1 {2} 4.4). Arguments for *PXFUNAME()* correspond to the arguments for *uname()*, as shown in Table 4.9.

Table 4.9—Arguments for *PXFUNAME()*

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
JUTSNAM E	name	IN	1.
IERROR	ret_value/errno	OUT	
1. Handle obtained from <i>PXFSTRUCTCREATE</i> ('utsname',...); see 8.3.1.			

The *PXFSTRUCTCREATE()* subroutine (see 8.3.1) with the string 'utsname' given as the *STRUCTNAME* argument shall be used to obtain a handle for an instance of the *utsname* structure as defined in POSIX.1 {2} 4.4. Each component access shall require one of the following structure-component manipulation subroutines (see 8.3.2):

```
SUBROUTINE PXFSTRGET (JUTSNAME, COMPNAM, SVALUE, ILEN, IERROR)
  INTEGER JUTSNAME, ILEN, IERROR
  CHARACTER*(*) SVALUE
```

where *JUTSNAME* is a handle and *COMPNAM* is a character expression which evaluates to one of the component names shown in Table 4.10.

Table 4.10—Components for *utsname* Structure

POSIX.1 Component	COMPNAM	Structure Procedures Used to Access
sysname	'sysname'	PXFSTRGET
nodename	'nodename'	PXFSTRGET
release	'release'	PXFSTRGET
version	'version'	PXFSTRGET
machine	'machine'	PXFSTRGET

4.4.1.3 Errors

POSIX.1 {2} does not specify any error conditions that are required to be detected for the *uname()* function. Upon successful completion of *PXFUNAME()*, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4.5 Time

4.5.1 Get System Time

Subroutine: *PXFTIME()*

4.5.1.1 Synopsis

```
SUBROUTINE PXFTIME (ITIME, IERROR)
  INTEGER ITIME, IERROR
```

4.5.1.2 Description

The value of time is always returned in the argument *ITIME*. No indirection or separate return value is used or necessary. Otherwise, the *PXFTIME()* subroutine shall provide the same functionality as the POSIX.1 {2} function *time()* (see POSIX.1 {2} 4.5). Arguments for *PXFTIME()* correspond to the arguments for *time()*, as shown in Table 4.11.

Table 4.11—Arguments for *PXFTIME()*

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
ITIME	*tloc	OUT	1.
IERROR	ret_value/errno	OUT	
1. Value may exceed the range of a signed integer; see 2.3.2.2.			

4.5.1.3 Errors

Possible error conditions for *PXFTIME()* are identical to those for the POSIX.1 {2} function *time()*. *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4.5.2 Get Process Times

Subroutine: *PXFTIMES()*

4.5.2.1 Synopsis

```
SUBROUTINE PXFTIMES (JTMS, ITIME, IERROR)
  INTEGER JTMS, ITIME, IERROR
```

4.5.2.2 Description

The *PXFTIMES()* subroutine shall provide the same functionality as the POSIX.1 {2} function *times()* (see POSIX.1 {2} 4.5). Arguments for *PXFTIMES* correspond to the arguments for *times()*, as shown in Table 4.12.

Table 4.12—Arguments for *PXFTIMES()*

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
JTMS	buffer	IN	1.
ITIME	ret_value	OUT	2.
IERROR	ret_value/errno	OUT	

1. Handle obtained from *PXFSTRUCTCREATE* ('tms',...); see 8.3.1.
2. Value may exceed the range of a signed integer; see 2.3.2.2.

The *PXFSTRUCTCREATE()* subroutine (see 8.3.1) with the string 'tms' given as the *STRUCTNAME* argument shall be used to obtain a handle for an instance of the *tms* structure as defined in POSIX.1 {2} 4.5. Each component access shall require one of the following structure-component manipulation subroutines (see 8.3.2):

```
SUBROUTINE PXFINTGET(JTMS, COMPNAM, IVALUE, IERROR)
  INTEGER JTMS, IVALUE, IERROR
```

where *JTMS* is a handle and *COMPNAM* is a character expression which evaluates to one of the component names shown in Table 4.13.

Table 4.13—Components for *tms* Structure

POSIX.1 Component	COMPNAM	Structure Procedures Used to Access
tms_utime	'tms_utime'	PXFINTGET
tms_stime	'tms_stime'	PXFINTGET
tms_cutime	'tms_cutime'	PXFINTGET
tms_cstime	'tms_cstime'	PXFINTGET

The value of the *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* components may exceed the range of a signed integer. See 2.3.2.2.

4.5.2.3 Errors

POSIX.1 {2} does not specify any error conditions that are required to be detected for the *times()* function. Upon successful completion of *PXF TIMES()*, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4.6 Environment Variables

4.6.1 Environment Access

Subroutines: *PXFGETENV()*, *PXFSETENV()*, *PXFCLEARENV()*

4.6.1.1 Synopsis

```
SUBROUTINE PXFGETENV (NAME, LENNAME, VALUE, LENVAL, IERROR)
CHARACTER*(*) NAME, VALUE
INTEGER LENNAME, LENVAL, IERROR
```

```
SUBROUTINE PXFSETENV (NAME, LENNAME, NEW, LENNEW, IOVERWRITE, IERROR)
CHARACTER*(*) NAME, NEW
INTEGER LENNAME, LENNEW, IOVERWRITE, IERROR
```

```
SUBROUTINE PXFCLEARENV (IERROR)
INTEGER IERROR
```

4.6.1.2 Description

The argument *VALUE* shall be a valid user-space character variable; the static return area provided by POSIX.1 {2} is not supported. Upon completion of *PXFGETENV()*, *VALUE* shall contain the value for the specified name *NAME*, and *LENVAL* shall contain the length of the value. If the specified variable is found but has no value, the value of *LENVAL* shall be set to zero and *VALUE* shall be filled with blanks. If the specified variable cannot be found, the condition *EINVAL* shall be returned in *IERROR*. Otherwise, the *PXFGETENV()* subroutine shall provide the same functionality as the POSIX.1 {2} function *getenv()* (see POSIX.1 {2} 4.6).

The *PXFSETENV()* subroutine shall search the environment list (see POSIX.1 {2} 2.6) for a string of the form *name=value*, where *name* is the contents of the character argument *NAME*. If no such string is present, *PXFSETENV()* shall add a string of the form *name=new* to the environment list, where *new* is the contents of the character argument *NEW*. Otherwise, if the *IOVERWRITE* argument is nonzero, *PXFSETENV()* either shall change the existing value to the contents of *NEW* or shall delete the string *name=value* and add the string *name=new*. The values assigned to the environment variables are restricted as specified in POSIX.1 {2} 2.6.

The *PXFCLEARENV()* subroutine shall clear the process environment. No environment variables are defined immediately after a call to *PXFCLEARENV()*.

Arguments for *PXFGETENV()* correspond to the arguments for *getenv()*, as shown in Table 4.14.

Table 4.14—Arguments for *PXFGETENV()*

FORTRAN Argument	POSIX.1 Argument	Intent	Notes
NAME	name	IN	
LENNAME	--	IN	Length of NAME; see 2.3.2.4
VALUE	ret_value	OUT	
LENVAL	--	OUT	Returned length of VALUE
IERROR	ret_value/errno	OUT	

4.6.1.3 Errors

POSIX.1 {2} does not specify any error conditions that are required to be detected for the *getenv()* function. Upon successful completion of *PXFGETENV()*, the argument *IERROR* shall be set to zero. If any of the following conditions occur, *PXFGETENV()* shall set the argument to the corresponding value. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

[EINVAL]	The variable <i>NAME</i> is not in the environment list.
[ETRUNC]	The declared length of the argument <i>VALUE</i> is insufficient to contain the string to be returned. (See 2.3.2.4.)

Upon successful completion of *PXFSETENV()*, the argument *IERROR* shall be set to zero. If any of the following conditions occur, *PXFSETENV()* shall set the argument to the corresponding value. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

[ENOMEM]	Not enough memory is available to create the necessary structures.
----------	--

Upon successful completion of *PXFLCEARENV()*, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4.7 Terminal Identification

4.7.1 Generate Terminal Pathname

Subroutine: *PXFCTERMID()*

4.7.1.1 Synopsis

```
SUBROUTINE PXFCTERMID (S, ILEN, IERROR)
CHARACTER*(*) S
INTEGER ILEN, IERROR
```

4.7.1.2 Description

The argument *S* shall be a valid user-space character variable; the static return area provided by POSIX.1 {2} is not supported, and the maximum length of the returned string indicated by the symbolic constant *L_ctermid* provided by

POSIX.1 {2} is not supported. The argument *ILEN* shall contain zero if the pathname that would refer to the controlling terminal cannot be determined or if *PXFCTERMID* is unsuccessful. If the length of the returned value is longer than the length of the passed character variable *S*, the return value shall be truncated.

Otherwise, the *PXFCTERMID*() subroutine shall provide the same functionality as the POSIX.1 {2} function *ctermid*() (see POSIX.1 {2} 4.7).

Upon completion, *S* shall contain a string that represents the controlling terminal for the current process, and *ILEN* shall contain the length of the string. Arguments for *PXFCTERMID*() correspond to the arguments for *ctermid*(), as shown in Table 4.15.

Table 4.15—Arguments for *PXFCTERMID*()

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
<i>S</i>	<i>s</i>	OUT	
<i>ILEN</i>	--	OUT	Length of <i>S</i> ; see 2.3.2.4
<i>IERROR</i>	<i>ret_value/errno</i>	OUT	

4.7.1.3 Errors

POSIX.1 {2} does not specify any error conditions that are required to be detected for the *ctermid*() function. Upon successful completion of *PXFCTERMID*(), the argument *IERROR* shall be set to zero. If any of the following conditions occur, *PXFCTERMID*() shall set the argument to the corresponding value. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

[ETRUNC] The declared length of the argument *S* is insufficient to contain the string to be returned.
(See 2.3.2.4.)

4.7.2 Determine Terminal Device Name

Subroutines: *PXFTTYNAME*(), *PXFISATTY*()

4.7.2.1 Synopsis

```
SUBROUTINE PXFTTYNAME (IFILDES, S, ILEN, IERROR)
INTEGER IFILDES, ILEN, IERROR
CHARACTER*(*) S
```

```
SUBROUTINE PXFISATTY (IFILDES, ISATTY, IERROR)
INTEGER IFILDES, IERROR
LOGICAL ISATTY
```

4.7.2.2 Description

PXFTTYNAME() and *PXFISATTY*() shall provide the same functionality as the corresponding POSIX.1 {2} functions *ttyname*() and *isatty*() (see POSIX.1 {2} 4.7). Upon return, the value of *ISATTY* shall be .TRUE. if *IFILDES* contains a valid file descriptor associated with a terminal. Otherwise, it shall be .FALSE..

Upon completion of *PXFTTYNAME()*, *S* shall contain the terminal pathname, and *ILEN* shall contain the length of the string. If the length of the returned value is longer than the length of the passed character variable *S*, the return value shall be truncated.

Arguments for *PXFTTYNAME()* and *PXFISATTY()* correspond to the arguments for *ttyname()* and *isatty()*, as shown in Table 4.16.

Table 4.16—Arguments for *PXFTTYNAME()* and *PXFISATTY()*

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
IFILDES	fildes	IN	
S	ret_value	OUT	
ILEN	--	OUT	Length of S; see 2.3.2.4
ISATTY	ret_value	OUT	
IERROR	ret_value/errno	OUT	

4.7.2.3 Errors

POSIX.1 {2} does not specify any error conditions that are required to be detected for the *ttyname()* function. Upon successful completion of *PXFTTYNAME()*, the argument *IERROR* shall be set to zero. If any of the following conditions occur, *PXFTTYNAME()* shall set the argument to the corresponding value. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

[ETRUNC]	The declared length of the argument <i>S</i> is insufficient to contain the string to be returned. (See 2.3.2.4.)
[EBADF]	<i>IFILDES</i> is not a valid file descriptor.

POSIX.1 {2} does not specify any error conditions that are required to be detected for the *isatty()* function. Upon successful completion of *PXFISATTY()*, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

4.8 Configurable System Variables

4.8.1 Get Configurable System Variables

Subroutine: *PXFSYSCONF()*

4.8.1.1 Synopsis

```
SUBROUTINE PXFSYSCONF (NAME, IVAL, IERROR)
  INTEGER NAME
  INTEGER IVAL, IERROR
```

4.8.1.2 Description

The *PXFSYSCONF()* subroutine shall provide the same functionality as the POSIX.1 {2} function *sysconf()* (see POSIX.1 {2} 4.8). *NAME* is an integer value representing a symbolic system variable. Values for *NAME* shall be obtained through calls to any of the *PXFCONST()* procedures (see 8.2).

Access to the special symbol {CLK_TCK} is not included since POSIX.1 {2} declares such access to be obsolescent.

Arguments for *PXFSYSCONF()* correspond to the arguments for *sysconf()*, as shown in Table 4.17.

Table 4.17—Arguments for *PXFSYSCONF()*

FORTTRAN Argument	POSIX.1 Argument	Intent	Notes
NAME	name	IN	
IVAL	ret_value	OUT	
IERROR	ret_value/errno	OUT	

4.8.1.3 Errors

Possible error conditions for *PXFSYSCONF()* are identical to those for the POSIX.1 {2} function *sysconf()*. *IERROR* shall be set to the corresponding nonzero value specified by the POSIX.1 {2} function. Upon successful completion, the argument *IERROR* shall be set to zero. *IERROR* may be set to a nonzero value to indicate error conditions that are not specified by POSIX.1 {2} and POSIX.9.

5. Files and Directories

5.1 Directories

5.1.1 Format of Directory Entries

The *PXFSTRUCTCREATE()* subroutine (see 8.3.1) with the string ‘dirent’ given as the *STRUCTNAME* argument shall be used to obtain a handle for an instance of the *dirent* structure as defined in POSIX.1 {2} 5.1. Each component access shall require one of the following structure-component manipulation subroutines (see 8.3.2):

```
SUBROUTINE PXFSTRGET(JDIRENT, COMPNAM, SVALUE, ILEN, IERROR)
INTEGER, JDIRENT, ILEN, IERROR
CHARACTER*(*) SVALUE
```

where *JDIRENT* is a handle and *COMPNAM* is a character expression which evaluates to one of the component names shown in Table 5.1.

Table 5.1—Components for *dirent* Structure

POSIX.1 Component	COMPNAM	Structure Procedures Used to Access
d_name	‘d_name’	PXFSTRGET