Application Programming Interface

for

PA-RISC Systems

PRECISION RISC ORGANIZATION
Printed in USA Draft: October 7, 1993
Internal Version #1.5

Legal Notices

The information contained in this document is subject to change without notice.

PRECISION RISC ORGANIZATION makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

Precision Risc Organization shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Precision Risc Organization assumes no responsibility for the use or reliability of software or equipment developed to this specification.

PRECISION RISC ORGANIZATION
19111 Pruneridge Avenue
Cupertino, California 95014 U.S.A.

Printing History

New editions of this manual will incorporate all material updated since the previous edition. The manual printing date indicates its current edition. The printing date changes when a new edition is printed. (Minor corrections which are incorporated at reprint do not cause the date to change.)

Internal Version #1.5 October 1993

Contents:

1

Introduction

The PRO *Application Programming Interface* (API) defines a structured set of source level interfaces to support the implementation of portable application software for PRO compliant systems. This specification is composed of industry standard APIs which provide an open and broadly supported development environment.

Purpose

This document provides a standard set of source level interfaces to support the development of application software that can be transported from one PRO API compliant platform to another at the source level. Proprietary specifications in the API are minimized to support the porting of application software source code. The goal of the PRO API is to support source — not object— code portability. The PRO API defines the programming interface; the PRO Application Binary Interface (ABI) defines the implementation methodology.

A major objective of the Precision Risc Organization is to provide a system environment that enables software portability. To support this open environment, PRO is establishing specifications for source and object level interfaces. These specifications are the PRO Application Program Interface and the PRO Application Binary Interface.

The PRO API and the PRO ABI are complementary specifications. The API specifies the entry points and interfaces (or procedures and data) at the source code level that will be supported by a system platform. Applications which are written to conform to the API will be portable at the *source code level* across system platforms that conform to the API; that is, they must be re-compiled on the new host system.

The Application *Binary* Interface (ABI) defines the interface between applications and system platforms, and the conventions of the interaction between the two.

Defined in the ABI are the binary format of applications, system entry points, certain system files & features, and the protocols of specified system functions.

The majority of the elements in the PRO API are existing industry standard APIs, such as POSIX 1003.1 and XPG/4, that are the product of standards organizations. Other standards that may be included are de facto standards or vendor standards that add value to the standardization of the source interface and have been approved by PRO.

One of the requirements of any standard is that it be possible to assess compliance. The PRO API provides a structure that allows systems vendors to state their compliance in an explicit yet simple manner through the use of layers and options. The current infrastructure is defined so that compliance can be stated simply as "PRO API Layer 2 Options P1, U2, N1" rather the more unwieldy POSIX.1, XPG4, OSF/AES, POSIX.2, ANSI C, Motif 1.2, X11R5, and ARPA Services.

The operating system services are defined in a layered manner to provide an increasing amount of system functionality with each successive layer. The options are defined to provide additional features that may be desired at any layer—such as languages or networking—but are not required by the system. The intention of this structure is to allow the system vendors the flexibility to define their products to address different markets. It is not the intention of the PRO API to dictate or imply specific product structures and for this reason, the number of layers is minimal. Layers are labelled with a number while options are labelled with one or more characters.

Definition of Terms

The API infrastructure is composed of *elements*, *layers* and *options*.

Element - Any API or portion thereof adopted by PRO for inclusion in the PRO API is an element. Any PRO-unique API feature is also an element.The element is the basic building block of the PRO API. Elements are combined according to the methodology described in this chapter.

Layer - A layer is comprised of elements. Layers are structured from the lowest (most minimal) to the highest (most complete). A given layer contains all of the elements defined for that layer plus all of the elements from any preceding layers.

For example, Layer 2 would define the elements that are listed as part of the definition of Layer 2. In addition, Layer 2 would also include all of the elements defined for Layers1.

Option - An option is an element or a combination of elements that may be included in a layer. Options are not requirements of a layer and may be added at any layer unless otherwise noted. For example, the FORTRAN language is not a required element; however, it is appropriate to allow FORTRAN to be included at any layer. This approach provides the flexibility to define products that are compliant with the PRO API thereby meeting the needs of different markets.

Organization

The organization of this API document reflects the structure of the API definition; first the layers are described then the options.

All of the APIs specified at a particular layer must be available in order for a platform to be API compliant at that layer. APIs may be implemented from higher layers at any time, however, an implementation will not be considered compliant to a particular layer unless *all* of the APIs specified for that layer are implemented. Required APIs are preceded by a ■ (bullet) symbol.

In a number of instances, the same standard has been adopted by multiple standards bodies. Where this is the case, the first instance of the standard is preceded by the bullet symbol; standards that are considered technically equivalent will then be listed, but will not be preceded by the bullet symbol. This approach is in recognition of the fact that the same standard may be known by different names or document numbers depending upon the issuing organization.

■ **Layer 1:** All APIs listed must be available on all system platforms. These APIs provide the base environment for a Layer 1 compliant implementation. The developer may selectively implement additional functionality that is not required by the base environment through the use of Options. Options are not required in order for the platform to be Layer 1 compliant.

■ **Layer 2:** Standards listed under this category include all of the standards specified for a Layer 1 implementation, as well as the standards listed for a Layer 2 compliant platform. As previously stated, additional functionality may be

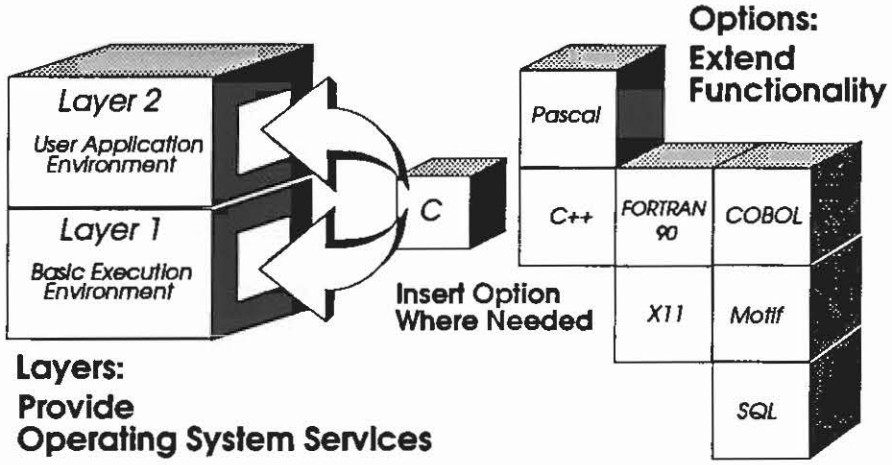
added by the developer through the implementation of Options. Options are not required in order for the platform to be Layer 2 compliant.

- **Options:** APIs listed in the Options section are not required. They are available for inclusion at the discretion of the developer in order to provide additional functionality. Options that are included must conform to the specifications listed in Chapter 4 of this document in order to be API compliant.

The options are divided into the following sections:

- Programming Languages and Compilers
- Graphics
- User Interface Services
- Interworking
- OSI/ISO Services
- ARPA Services
- Berkeley Services
- ONC Services
- Transports
- Transport Interfaces
- Network Management
- Communications/ I/O Portability Environments
- Object Oriented Technology
- Distributed Services
- Data Management Services

The following is an illustration of how options may be applied at any layer; for example C language may be implemented at either Layer 1 or Layer 2. Please note that this illustration shows only a small number of the options available.



PRO API MODEL

Compliance

A compliant platform is one that provides the required source interfaces for specific layers and options as defined in this API document. PRO will not be providing test suites for the API, since this API is based on existing industry standards and test suites are generally available based upon those standards.

Application software will not be certified as PRO API compliant. Since few ISVs permit circulation of their source code outside of their company, there is little value in providing certification at that level. Also, it is to the advantage of ISVs to write the source to match the API. Applications are distributed in compiled formats (typically linkable or executable) and PRO API certification for an application

would be of little consequence to a user. PRO ABI certification, however, will be available.

Platform developers are free to provide any additional APIs on any PRO API compliant system platforms that they wish. ISVs are free to use any of such additional APIs, but, the ability to compile source code on other compliant platforms will be limited by such use.

Standards Precedence and Conflict Resolution

In cases where standards and specifications of the PRO API conflict, the conflict resolution model specified by the OSF AES will be applied. The following rules summarize the model:

In cases where technically equivalent standards exist, the relevant *international* standard has the highest precedence. Following in precedence are (from highest to lowest):

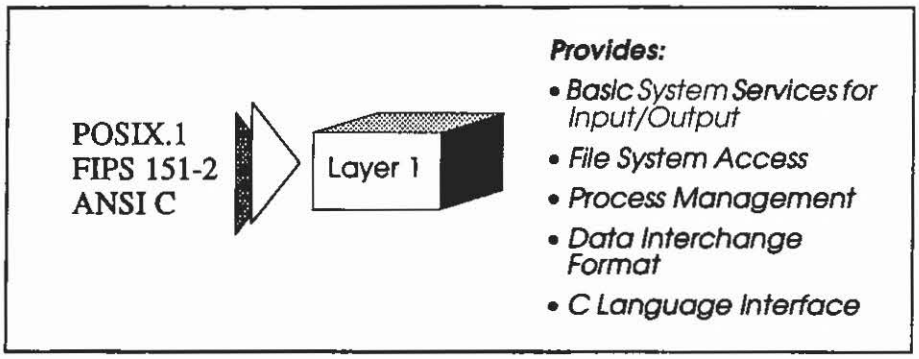
- Internationally Recognized Voluntary Standards (for example, ISO standards).
- Accredited National Standards (for example, ANSI standards).
- Accredited Voluntary Standards (for example, IEEE standards).
- Industry Specifications (for example, the X/Open Portability Guide).
- Vendor Specifications (for example, the OSF AES).
- Implementation (for example, BSD 4.3).

Conflicts between mandatory interface specifications shall be resolved according to the following precedence hierarchy (from highest to lowest):

- POSIX 1003.1
- ANSI C
- XPG4
- OSF AES

2

Layer 1



Layer 1: Basic Execution Environment

Base Layer:

This section covers the API standards necessary for a Layer 1 PRO API implementation.

Required:

- ISO/IEC 9945-1:1990 Information Technology—Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API) [C Language] (Commonly known as POSIX.1) (adopted IEEE/ANSI 1003.1-1990)

PRO considers the following to be a technical equivalent to the standard listed above:

IEEE/ANSI 1003.1-1990 Information Technology—Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API) [C Language] (Commonly known as POSIX.1)

- NIST FIPS 151-2 Portable Operating System Interface (POSIX)—System Application Program Interface [C Language] 1993 May 12. (Adopted ISO/IEC 9945-1:1990)
- ISO/IEC 9899:1990 (E) Programming languages - C (only as required by Section 8 of the POSIX.1 standard)
- The following additional entry points are required by the ABI:

  _filbuf _flsbuf

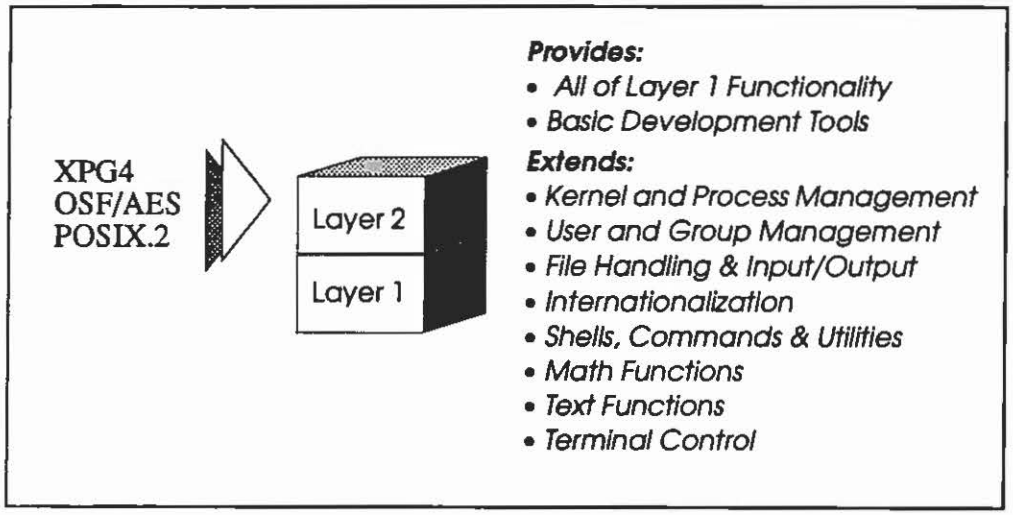
Rationale:

POSIX.1 is an international standard that provides a common Unix-like environment as well as operating system services through the use of a C language interface. In order to best support the application developer, the standard defines the external characteristics that are of importance to the developer rather than defining the internal constructs supporting the facilities and services. The scope of the standard encompasses terminology and general requirements, definitions for system service interfaces and subroutines, language-specific system services for the C programming language, and interface issues, including portability, error handling and recovery. POSIX.1 does not specify shells or their associated commands, networking protocols or their associated system calls, graphic interfaces, database management system interfaces, record I/O, binary code portability, system configuration or resource availability, nor the behavior of system services on systems supporting concurrency within a single process. Future revisions of POSIX.1 are expected to provide bindings to other programming languages in addition to C.

The PRO Application Binary Interface (ABI) requires entry points in addition to those specified by the standards for Layer 1. These entry points are primarily for system platform implementations, and not typically used by application developers.

3

Layer 2



Layer 2: User Application Environment

Base Layer:

This section covers the API standards necessary for a Layer 2 PRO API implementation.

Required:

- All of the standards listed in Layer 1, plus:
- X/Open XPG4 Base, plus selected feature groups.

  Components contained in the Base Profile:

  XPG4 Internationalized System Calls and Libraries
  XPG4 Commands and Utilities
  XPG4 C Language

The following XPG4 feature groups are required:
Shared Memory
POSIX2 C-language Binding.

- OSF Application Environment Specification (AES) Operating System Programming Interfaces

- ISO/IEC 9945-2:1992 Information technology - Portable Operating System Interface (POSIX) Part 2: Shells and Utilities (Adopted IEEE 1003.2) (Commonly known as POSIX.2)

PRO considers the following to be a technical equivalent to the preceding standard:

IEEE 1003.2 Information Technology—Portable Operating System Interface (POSIX)—Part 2: Shells and Utilities

- The following additional entry points are required by the ABI:

ioctl	ptrace	ptsname
sbrk	shl_definesym	shl_findsym
shl_get	shl_gethandle	shl_getsymbols
shl_load	shl_unload	

Rationale:

XPG4 adds functionality to a Layer 1 implementation through extensions to the POSIX.1 standard. X/Open has developed a Common Applications Environment (CAE) as the base for its computing platform; XPG4 defines the interfaces between the CAE components. The base is comprised of commands and utilities, system calls, and C language. XPG4 provides for localization through the use of internationalized system calls and libraries.

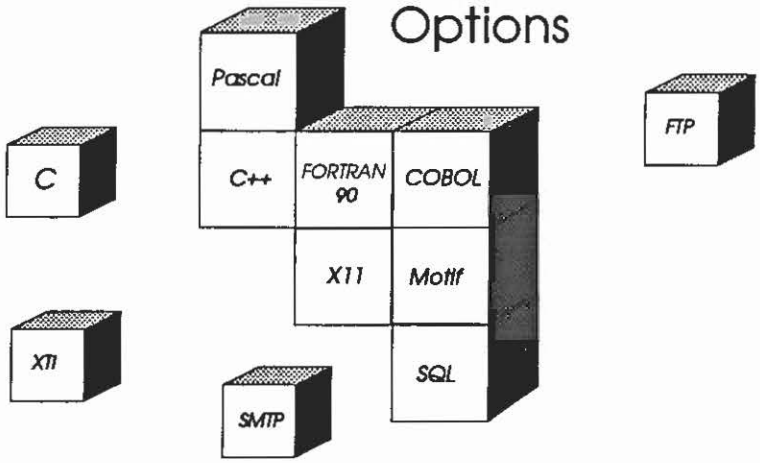
OSF/AES is also based on the POSIX.1 specification and further extends the functionality of a Layer 1 implementation.

In the event that a conflict between standards arises, POSIX.1 shall take precedence over XPG4 and AES.

The PRO Application Binary Interface (ABI) requires entry points in addition to the standards specified for Layer 2.

4

Options



Options allow developers the flexibility to provide additional features and functions to the base operating system service layers as required for support of their applications. Options are not required by the system for PRO compliance; however, if an option is declared for PRO compliance, all of the required elements listed for that option must be implemented. PRO does not restrict the implementation of options to specific layers unless otherwise noted.

Programming Languages & Compilers:

This section covers the API standards for languages and compilers.

Rationale:

While POSIX.1 specifies C language bindings, developers may choose to implement other languages. The following are the API standards necessary to implement the associated programming languages.

P1: C

- ISO/IEC 9899:1990 Programming languages - C
  (Except for addenda, same content as ANSI X3.159- 1989)

PRO considers the following to be technically equivalent to the preceding standard:

ANSI X3.159-1989 Programming Language - C

NIST FIPS 160 C (Adopted ANSI X3.159-1989)

P2: C++

- No standard at present.

Recommendation:

Conform to USL C++ version 3.0

Fortran

Rationale:

While Fortran 90 is the more recent of the two standards and the direction that many developers are heading, demands of industry and government currently dictate that FORTRAN 77 be retained. It should be noted that there is no longer a valid ISO standard in the United States for FORTRAN 77, it has been replaced by the Fortran 90 standard. Both FORTRAN 77 and Fortran 90 standards are included at this time and should be referenced as P3A for FORTRAN 77 and P3B for Fortran 90, respectively.

P3A: FORTRAN 77

- ANSI X3.9-1978 FORTRAN 77

PRO considers the following to be technically equivalent to the preceding standard:

NIST FIPS 69-1 FORTRAN (Adopted ANSI X3.9-1978)

Recommendation for United States:

MIL-STD-1753 FORTRAN (DoD Supplement to ANSI X3.9-1978)

P3B: Fortran 90

- ISO/IEC 1539:1991 Information Technology - Programming languages - FORTRAN

PRO considers the following to be technically equivalent to the preceding standard:

ANSI X3.198-1992 Fortran 90

P4: Pascal

- ISO 7185:1990 Information technology - Programming languages - Pascal

PRO considers the following to be technically equivalent to the preceding standard:

ANSI/IEEE770X3.97-1983 (R1990) Pascal Computer Programming Language

NIST FIPS 109 Pascal (Adopted ANSI/IEEE770X3.97-1983 (R1990))

P5: COBOL

- ISO 1989:1985 Programming languages - COBOL (Endorsement of ANSI X3.23-1985)
- ISO/IEC 1989 Ammendment 1 Programming languages - Intrinsic Function Module for COBOL (Endorsement of ANSI X3.23a-1989)

PRO considers the following to be technically equivalent to the preceding standards:

ANSI X3.23-1985 (R 1991) Programming Language - COBOL

ANSI X3.23a-1989 (R 1991) Programming Language - Intrinsic Function Module for COBOL

NIST FIPS 21-3 COBOL (Adopted ANSI X3.23-1985 and ANSI X3.23A-1989)

P6: ADA

- ISO 8652:1987 Programming languages - ADA (Adopted ANSI/MIL-STD-1815A-1983)

PRO considers the following to be technically equivalent to the preceding standard:

ANSI/MIL-STD-1815A ADA Programming Language

NIST FIPS 119 ADA (Adopted ANSI/MIL-STD-1815A-1983)

P7: LISP

- No standard at present.

Recommendation:

Guy Steele's Common Lisp: The Language, 2nd Edition

P8: Assembly

- HP 9000 Computer Systems Assembly Language Reference Manual HP Part No. 92432-90001

Graphics:

This section covers the API standards for Graphics Services.

G1: GKS

- ISO 7942:1985 (Amendment 1 1991) Computer graphics - Graphical Kernel System (GKS) Functional Description
- ISO 8651-1: 1988 Information processing systems - Computer graphics - Graphical Kernel System (GKS) language bindings - Part 1 : FORTRAN
- ISO 8651-2: 1988 Information processing systems - Computer graphics - Graphical Kernel System (GKS) language bindings - Part 2 : Pascal
- ISO 8651-3: 1988 Information processing systems - Computer graphics - Graphical Kernel System (GKS) language bindings - Part 3 : Ada
- ISO 8651-4: 1991 Information technology - Computer graphics - Graphical Kernel System (GKS) language bindings - Part 4 : C

PRO considers the following to be technically equivalent to the standards listed above:

ANSI X3.124-1985 (R 1991) Graphical Kernel System (GKS) Functional Description (Includes ANSI X3.124.1-1985)

ANSI X3.124.1-1985 (R 1991) Graphical Kernel System (GKS) FORTRAN Binding

ANSI X3.124.2-1988 Information Systems Computer Graphics - Graphical Kernel System (GKS) Pascal Binding

ANSI X3.124.3-1989 Computer Graphics - Graphical Kernel System (GKS), Ada Binding

NIST FIPS 120-1 Graphical Kernel System
(Adopted ANSI GKS X3.124-1985, ANSI X3.124.1-1985, ANSI X3.124.2-1988, & ANSI X3.124.3-1989)

G2: PHIGS

- ISO/IEC 9592-1:1989 (Part 1) Information processing systems - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS) - Part 1: Functional description
- ISO/IEC 9592-2:1989 (Part 2) Information processing systems - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS) - Part 2: Archive file format
- ISO/IEC 9592-3:1989 (Part 3) Information processing systems - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS) - Part 3: Clear-text encoding of archive file
- ISO/IEC 9593-1:1991 (Part 1) Information technology - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS) language bindings - Part 1: FORTRAN
- ISO 9593-4:1991 (Part 4) Information technology - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS) language bindings - Part 4: C

G3: PHIGS PLUS

- ISO/IEC 9592-4:1992 (Part 4) Information technology - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS) - Part 4: Plus Lumière und Surfaces, PHIGS PLUS
- ISO/IEC 9593-1:1990 (Part 1) Information technology - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS) language bindings - Part 1: FORTRAN
- ISO 9593-4:1991 (Part 4) Information technology - Computer graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS) language bindings - Part 4: C

User Interface Services:

This section covers the API standards for User Interface Services.

U1: X Window System

- MIT X Consortium Standard: X Window System Version 11, Release 5

U2: Motif

- OSF/Motif Version 1.2

The following is required for Motif:

- MIT X Consortium Standard: X Window System, Version 11, Release 5

U3: PEX (PHIGS Extension to the X Window System)

- MIT X Consortium Standard: X Window System, Version 11, Release 5
- MIT X Consortium PEX Protocol Specification Version 5.1
- MIT X Consortium PEXlib Specification and C Language Binding, Version 5.1
- MIT X Consortium PEX Protocol Encoding Version 5.1

Interworking:

The various standards bodies each define their own methods for implementing the network services listed on the following pages. The terms described below are to provide a context for understanding the matrix on the next page:

Links: The means for networks to physically transmit data from one point to another point. Included are both the hardware (interface controllers and cables) and the software device drivers. The software at this level defines how a network converts the raw data into packets that may then be transmitted.

Transports: Network transports provide the means for networks to route the data packets as well as ensuring the integrity of the data. Transports often provide programmatic access to one or more layers of a network model. Transports include OSI, TCP/IP and UDP/IP.

Services: Services, such as ARPA, NFS, and X.400, exist at the highest layers of networking models. Both end users and application developers directly access network services.

The Networking Services Matrix on the following page, shows what specific network services are available for each major group of protocols, as well as the network transport supporting those protocols. The leftmost column breaks down the network services into a list of the common or generic network service types, while the remaining columns list each network service, grouped under the appropriate network protocol. In some instances, the same service may appear more than once, or in more than one column, as some protocols may provide more than one service.

Network Services Matrix

Generic Network Services	STANDARD NETWORK PROTOCOLS					
	Services Using the OSI Network Transport	Services Using the TCP/UDP/IP Network Transport				
		ARPA	BSD	ONC	DCE	
Virtual Terminal	VTP	TELNET	rlogin			Services
File Transfer	FTAM	FTP, TFTP	rcp			
Remote File Access	FTAM	NCS		NFS	DFS	
Remote Database Access	RDA					
Remote Peripheral Access	FTAM					
Distributed File Systems		NCS		NFS	DFS	
Messaging	X.400	SMTP				
Directory Services	X.500		DNS	NIS	CDS, GDS	
Remote Procedure Calls (RPCs)	MMS	NCS		RPC/ XDR	DCE RPC	Service/ Development
Remote Process Management	MMS		rsh	REX		
Network Transport APIs (Interprocess Communication)	XTI	XTI, BSD Sockets				Transport APIs
Network Management	CMIP	SNMP				
Security					DCE (Kerberos)	

OSI/ISO Services:

N1A: X400

XPG4 X.400 Gateway and Message Access

- API to Electronic Mail (X.400)
  (X/Open CAE Specification: C191)
- OSI - Abstract - Data Manipulation API (XOM) (Was OSI Object Management)
  (X/Open CAE Specification: C180)

N1B: X.500

XPG4 Directory Access

- API to Directory Services (XDS)
  (X/Open CAE Spcification: C190)
- OSI - Abstract - Data Manipulation API (XOM) (Was OSI Object Management)
  (Document Number: C180)

N1C: FTAM

OSI File Transfer and Access Management

- MAP/TOP 3.0 standard

N1D: MMS

OSI/ISO Manufacturing Message System

- as specified by MMS-I standard interface specification, part of the MAP/TOP 3.0 specification

ARPA Services:

N2A: SMTP (Simple Mail Transfer Protocol)

- IAB Standard 10 (see RFC 821)

  Also see: MIL-STD-1781 Simple Mail Transfer Protocol

The following is required for ARPA Services:

- TCP/IP Network Transport

N2B: FTP (File Transfer Protocol)

- IAB Standard 9 (see RFC 959)

  Also see: MIL-STD-1780 File Transfer Protocol

The following is required for ARPA Services:

- TCP/IP Network Transport

N2C: TELNET

- IAB Standard 8 (see RFC 854, 855)

  Also see: MIL-STD-1782 TELNET

The following is required for ARPA Services:

- TCP/IP Network Transport

N2D: TFTP

- IAB Standard 33 (see RFC 1350)

The following is required for ARPA Services:

- TCP/IP Network Transport

Berkeley Services:

N3A: BSD (Berkeley Software Distribution) 4.3

The following Berkeley Services must be supported:

- rlogin (Remote Login)
- rcp (Remote Copy)
- rsh (Remote Shell)
- BIND (Berkeley Internet Name Domain - network information lookup service)

The following are required for ARPA/Berkeley Services:

- TCP/IP Network Transport
- BSD Sockets

ONC Services:

N4A: RPC/XDR

- RFC 1057 RPC: Remote Procedure Call Protocol specification: Version 2. Sun Microsystems, Inc. 1988 June
- RFC 1014 XDR: External Data Representation standard. Sun Microsystems, Inc. 1987 June

N4B: NFS

- NFS 4.2 Network File System Protocol specification. Sun Microsystems, Inc.

Transports:

N5A: TCP/IP

PRO recommends TCP/IP protocol for the following services: ARPA/ Berkeley (BSD).

- IAB Standard 7 (see RFC 793) Transmission Control Protocol. Postel, J.B. 1981 September

  See also: MIL-STD-1778 Transmission Control Protocol

- IAB Standard 5 (see RFC 791) Internet Protocol. Postel, J.B. 1981 September. (Obsoletes RFC 760) As ammended by:

  RFC 950 Internet standard subnetting procedure. Mogul, J.C.; Postel, J.B. 1985 August

  RFC 922 Broadcasting Internet datagrams in the presence of subnets. Mogul, J.C. 1984 October

  RFC 919 Broadcasting Internet datagrams. Mogul, J.C. 1984 October

  See also: MIL-STD-1777 Transmission Control Protocol

- IAB Standard 3 (see RFC 1122) Requirements for Internet hosts - communication layers. Braden, R.T., ed. 1989 October

- IAB Standard 3 (see RFC 1123) Requirements for Internet hosts - application and support. Braden, R.T., ed. 1989 October

N5B: UDP/IP

- IAB Standard 6 (see RFC 768) User Datagram Control Protocol. Postel, J.B. 1980 August 28

- IAB Standard 5 (see RFC 791) Internet Protocol. Postel, J.B. 1981 September. (Obsoletes RFC 760) As ammended by:

  RFC 950 Internet standard subnetting procedure. Mogul, J.C.; Postel, J.B. 1985 August

  RFC 922 Broadcasting Internet datagrams in the presence of subnets. Mogul, J.C. 1984 October

  RFC 919 Broadcasting Internet datagrams. Mogul, J.C. 1984 October

  See also: MIL-STD-1777 Transmission Control Protocol

Transport Interfaces:

Note:

The particular network transports that are accessible via these APIs will be specific to each vendor's system.

N6A: BSD SOCKETS:

- Berkeley IPC Programmer's Guide (4.3 BSD Sockets)

N6B: XTI

- X/Open Transport Interface (XTI)
  (X/Open CAE Specification: C196)

Network Management:

N7A: CMIP:

- ISO/IEC 9596-1:1991 Information technology - Open Systems Interconnection - Common management information protocol - Part 1: Specification. Technical Corrigendum 1:1992

N7B: SNMP

- IAB Standard15 (see RFC 1157) A Simple Network Management Protocol, Case, J., M. Fedor, M. Schoffstall, and J. Davin,

Communications / I/O Software Portability Environments:

A communications / I/O portability environment provides a generalized framework and toolset for the development of portable communications and I/O software.

C1: STREAMS Environment

The STREAMS environment includes an associated API, however the specific communications or I/O software that is accessible via this API will be specific to each vendor's system.

The definition of "STREAMS environment" as used in this document does not include facilities for STREAMS-based terminal or pseudo-terminal subsystems.

- USL System V Interface Definition, Third Edition (SVID3)

Object Oriented Technology:

O1: Common Object Request Broker (CORBA)

- OMG CORBA 1.1

PRO considers the following to be technically equivalent to the preceding standard:

X/Open Object Request Broker (X/Open C207)

Distributed Services:

S1: Distributed Computing Environment (DCE)

- OSF Application Environment Specification Distributed Computing (AES/DC) Volume, Revision A

Data Management Services:

This section covers the API standards for Data Management Services.

Rationale:

The SQL92 standard specifies three levels of implementation: Entry, Intermediate and Full. The Entry level is the complete SQL database language and consists of the existing SQL89 standard with integrity features, as well as the Embedded SQL standard. The Intermediate level adds features that are not yet commonly available in products, and the Full level adds many more complex features.

Entry Level is the PRO supported level of conformance.

D1: SQL (Entry Level)

- ISO/IEC 9075:1992 Information processing systems - Database Language SQL with integrity enhancement (Level 1)

PRO considers the following to be technically equivalent to the preceding standard:

ANSI X3.135-1992 Database Language - SQL with Integrity Enhancement

- X/Open Structured Query Language (SQL) (X/Open Specification: C201)

5

Test Suites

The following list of validation suites is provided only as a reference guide to industry, de facto, and commercially available verification suites for developers desiring to test their applications at a source code level. PRO makes no recommendations or warranties with regard to the following list. Developers should be aware that source code level testing is not an assurance of binary portability nor are these suites intended as a substitute for the PRO Conformance Environment testing.

LAYER 1:

POSIX.1

- The National Institute of Standards and Technology has established a conformance testing program for FIPS 151-1 and FIPS 151-2 through its Computer Systems Laboratory. Also part of NIST is the National Voluntary Laboratory Accreditation Program (NVLAP), responsible for accrediting testing laboratories. Testing services are provided through these Accredited POSIX Testing Laboratories (APTLs) who forward final test results to NIST/CSL in order to obtain a Certificate of Validation. To obtain a list of testing laboratories, contact NIST.
- IEEE Standard for Information Technology - Test Methods for Measuring Conformance to POSIX. (Order from IEEE #1003.3-1991)

FIPS 151-1

- NIST-PCTS:151-1 POSIX Conformance Test Suite (PCTS) (For testing conformance to 151-1 and reference standard IEEE 2003.1). (Order from NTIS #PB90-500919/CAU)

FIPS 151-2

- NIST-PCTS:151-2 POSIX Conformance Test Suite (PCTS) (For testing conformance to 151-2.) Tests conformance based on the test method specifications of POSIX.3.1-1992 and the additional specific requirements "a - p" of FIPS 151-2. (Order from NIST/CSL)

LAYER 2:

X/Open XPG4

- X/Open Verification Suite, Version 4, for testing XPG4 (Order from X/Open, #VSX4)

AES

- OSF VSE Validation Test Suite (Order from OSF, #VSE)

POSIX.2

- Perennial POSIX 1003.2 Shell and Utilities Validation Suite Early Access Version 1.3 (Order from Perennial, #PVS-2)

OPTIONS:

PROGRAMMING LANGUAGES AND COMPILERS:

ISO C

- The Plum Hall Validation Suite for C (Order from Plum Hall)
- X/Open VSX4 C Language test report (Order from X/Open)
- Perennial C Validation Suite (Order from Perennial, #CVS-A)

ANSI C

- The Plum Hall Validation Suite for C (Order from Plum Hall)
- Perennial ANSI C Validation Suite, Version 4 (Order from Perennial, #ACVS)
- C Validation Suite (Order from MetaWare)

C++

- Suite++: The Plum C++ Validation Suite (Order from Plum Hall)
- Perennial/USL C++ Verification Suite, Version 3.1 (Order from Perennial, #C++VS)

FORTRAN 77

- NIST FORTRAN Compiler Validation (FCVS78) (Order from NTIS, #PB85-226736)

Pascal

- BSI Pascal Validation System (PVS) (Order from the British Standards Institute)

COBOL

- NIST COBOL Compiler Validation System (CCVS85) (Order from NTIS, #PB91-508002/CAU)

Ada

- NIST Ada Compiler Validation (Order from NTIS, #ADA212548)

GRAPHICS:

GKS

- NIST GKS Validation Test Suite (Available through NIST)

USER INTERFACE SERVICES:

X protocol and Xlib:

- "Unisoft" test suite. (Order from X Consortium)

Motif

- Motif Validation Test Suite (VTS) (Order from OSF)
- Motif QA Test Suite (QATS) (Order from OSF)

INTERWORKING:

BSD 4.3

- Perennial BSD 4.3 Validation Suite, Version 1.5 (Order from Perennial, #UVS-E) Note: Developers should note that this suite tests more than just Sockets.

OSF Distributed Computing Environment (DCE)

- OSF DCE RPC Validation Test Suite (VTS) 1.0 (Order from OSF)

DATA MANAGEMENT SERVICES:

SQL

- NIST SQL Validation System Version 2.0.2. (Order from NIST)

TEST SUITE SOURCES:

BSI

British Standards Institute
Software Engineering Department
BSI Quality Assurance
P.O. Box 375
Milton Keynes
MK14 6LL
England

Telephone: +44-908-220908
Fax: +44-908-220671

MetaWare

MetaWare Inc.,
2161 Delaware Avenue
Santa Cruz, CA 95060-5706
USA

Telephone: +1(408) 429-6382
Fax: +1(408) 429-9273

NIST

For **POSIX:**

National Institute of Standards and Technology (NIST)
Computer Systems Laboratory
POSIX Certification Authority
Building 225, Room B266
Gaithersburg, MD 20899
USA

Telephone: +1(301) 975-3295
Fax: +1(301) 590-0932

For **SQL**:

National Institute of Standards and Technology (NIST)
Computer Systems Laboratory
Database and Graphics Group
Building 225, Room A266
Gaithersburg, MD 20899
USA

Telephone: +1(301) 975-3258
Telephone: +1(301) 975-3263
Fax: +1(301) 590-0932

For **GKS (Fortran):**

Ms. Susan Sherrick
National Institute of Standards and Technology (NIST)
Computer Systems Laboratory
Building 225, Room A266
Gaithersburg, MD 20899
USA

Telephone: +1(301) 975-3274

NTIS

For **COBOL, Fortran, and Ada:**
National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161
USA

Telephone: +1(703) 487-4650
Fax: +1(703) 321-8547

Perennial

Perennial
4699 Old Ironsides Drive, Suite 210
Santa Clara, CA 95054
USA

Telephone: +1(408) 748-2900
Fax: +1(408) 748-2909

Plum Hall

Plum Hall, Inc.
P.O. Box 44610
Kamuela, HI 96743
USA

Telephone: +1(808) 882-1255
Fax: +1(808) 882-1556

OSF

OSF Direct Channels
11 Cambridge Center
Cambridge, MA 02142
USA

Telephone: +1(617) 621-7300

X

X Consortium
One Memorial Drive
P.O. Box 546545
Cambridge, MA 02142-0004
USA

Telephone: +1(617) 374-1000

X/Open

X/Open Company Limited
Apex Plaza, Forbury Road
Reading
Berkshire, RG1 3BD,
United Kingdom

Telephone: +44 734 508311
Fax: +44 734 500110

X/Open Company Limited
1010 El Camino Real, Suite 380
Menlo Park, CA 94025
USA

Telephone: +1(415) 323-7992
Fax: +1(415) 323-8204

X/Open Company Ltd.
Karufuru-Kanda Bldg, 9F
1-2-1, Kanda Suda-cho
Chiyoda-ku, Tokyo 101
Japan

Telephone: +81 3 3251 8321
Fax: +81 3 3251 8376

A

Glossary

ANSI American National Standards Institute.

ARPA Advanced Research Project Agency, a standards-setting agency of the United States Department of Defense. Refers to a layered protocol suite for data communications.

CAE Common Applications Environment. The X/Open standards are known as CAE and are defined in the X/Open Portability Guide (XPG). The purpose of the CAE is to ensure application portability between systems that are CAE compliant.

CORBA Common Object Request Broker (CORBA). A standard for distributed object management solutions developed by the Object Management Group (OMG).

DCE OSF's Distributed Computing Environment. A comprehensive, integrated set of operating system and network independent services that support the development, use and maintenance of distributed applications.

FIPS Federal Information Processing Standards, written by NIST.

GKS Graphics Kernel System, a set of basic functions for computer graphics programming. The GKS standard allows graphics application programs to be easily transported between installations and aids graphics applications programmers in understanding and using graphics facilities.

IAB Internet Activities Board (IAB) is the coordinating committee for Internet design, engineering and management. The IAB manages the RFC process, and sets Internet standards.

IEC International Electrotechnical Commission.

IEEE Institute of Electrical and Electronics Engineers.

ISO International Organization for Standardization, a specialized international agency for standardization, at present comprising the national standards bodies of 91 countries. The object of ISO is to promote the development of standardization and related world activities with a view to facilitating international exchange of goods and services and to developing cooperation in the sphere of intellectual, scientific, technological, and economic activity. The results of ISO technical work are published as international standards.

ISO/IEC A joint technical committee made up of ISO and IEC members.

NIST National Institute of Standards and Technology. Formerly known as the National Bureau of Standards, NIST is an accredited ANSI standards making organization. The Federal Information Processing Standards (FIPS) written by NIST specify the requirements that may be used by federal agencies in procuring equipment and software.

OMG Object Management Group; a non-profit corporation. This group developed the Common Object Request Broker (CORBA) standard for distributed object management solutions.

ONC SunSoft's Open Network Computing Environment.

OSF Open Software Foundation; a not-for-profit, industry-supported research and development organization. It was established in order to define source code reference implementations and

specifications, develop a leading operating system, and promote a portable applications environment.

OSI Open Systems Interconnection, the ISO seven-layer architectural model for data communications networks.

PEX PEX is a protocol extension to the X Window System protocol that provides direct support for 3D graphics in the X environment. It was originally designed to efficiently support 3D graphics standard APIs such as PHIGS, PHIGS Plus, and GKS-3D. Recent work has shifted toward extending the capabilities to broaden support for other 3D APIs.

PHIGS Programmers Hierarchical Interactive Graphics Standard. A library of routines that enable applications to display either two-dimensional or three-dimensional world-coordinate data. An extension to PHIGS, called PHIGS PLUS, also provides a definition for lighting and shading.

POSIX 1003.2 POSIX.2 provides shell and tool facilitates for developers of portable applications. It specifies a shell command language based upon the System V shell, including some of the newer features of the Korn Shell, and a set of "tools" or commands. This standard also incorporates POSIX 1003.2a, commonly referred to as the User Portability Extensions (UPE).

RFC Request For Comments (RFCs) are documents maintained by the Internet Architecture Board (IAB). Those that are approved define standards for the Internet protocol suite. While all standards are published as RFCs, not all qualify as standards. The IAB tracks the status of RFCs.

USL Unix System Laboratory. USL is responsible for Unix System V operating system development and licensing. Originally under the control of AT&T's Bell Labs research subsidiary, control of System V was transferred to USL, and later sold to Novell.

X/Open Founded in 1984, X/Open is an independent, nonprofit consortium of international systems vendors working to specify the open, vendor-independent Common Applications Environment (CAE). Specification of the CAE is achieved through cooperation with users, independent software and hardware vendors, and standards organizations.

XPG4 The X/Open Operating System Interface (XSI) Common Application Environment (CAE) consists of the following:

- The XPG4 binder containing profiles, component definitions and branding information
- System Interfaces & Headers, Issue 4 (the XBD specification)
- Commands & Utilities, Issue 4 (the XCU specification)
- System Interface Definitions, Issue 4(the XSH specification)

B

Emerging Standards

This section lists APIs that are currently under development and are being considered by the Precision Risc Organization (PRO) for inclusion in this API. Until the standard is completed and has been accepted by PRO, it is not part of this API document. This information is included for information only and does not represent a guarantee that every standard listed here will become part of this API.

Operating System Services:

- IEEE POSIX P1003.4a (Threads)
- IEEE POSIX P1003.4b (Additional real-time extensions)
- IEEE POSIX P1003.6 (Security)
- IEEE POSIX P1003.7.1 (Printer Administration)
- IEEE POSIX P1003.7.2 (Software Administration)
- IEEE POSIX P1003.8 (Remote File Access)
- IEEE POSIX P1003.15 (Batch Queuing Extensions)

Interworking:

- X/Open XAP (ACSE/Presentation API)

Languages:

- ANSI Lisp (Under development by X3J13 committee)
- ISO C++ (Under development by ISO WG21 group)
- ANSI C++ (Under development by ANSI X3J16)

Data Management

- NIST FIPS 127-2 SQL

C

Standards Under Consideration

This section lists APIs that are currently under consideration by the Precision Risc Organization (PRO) for inclusion in this API document. This information is included for information only and does not represent a guarantee that every standard listed here will become part of this API.

- OSF Distributed Management Environment (DME)
- Protocols for X/Open Interworking: XNFS
- IEEE POSIX P1003.4 (Real-Time Extensions to 1003.1)
- X/Open Common Desktop Environment

D

Standards Sources

This section provides a partial list of the addresses of the standards organizations and companies that were mentioned in this API document.

ANSI

American National Standards Institute, Inc.
Attn: Customer Service
11 West 42nd Street
New York, NY 10036
USA

IEEE

IEEE Customer Service Department
445 Hoes Lane
P.O. Box 1331
Piscataway, New Jersey 08855-1331
USA

ISO

International Organization for Standardization/
International Electrotechnical Commission
1, rue de Varembé
Case postale 56
CH-1211
Genève 20
Switzerland/Suisse

MAP/TOP

Corporation for Open Systems
1750 Old Meadow Road, Suite 400
McLean, VA 22102
USA

MIL-STD
Department of the Navy
Naval Publications and Forms Center, Code 3015
5801 Tabor Avenue
Philadelphia, PA 19120-5099
USA

NTIS
(For NIST / FIPS publications)
National Technical Information Service
Springfield, Virginia 22161
USA

OMG
Object Management Group, Inc.
Framingham Corporate Center
492 Old Connecticut Path
Framingham, MA 01701
USA

OSF
Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142
USA

RFCs
DDN Network Information Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
USA

SUN
Sun Microsystems
2550 Garcia Avenue
Mountain View, CA 94043
USA

USL
UNIX System Laboratory
190 River Road
Summit, NJ 07901
USA

X Consortium
X Consortium
One Memorial Drive
P.O. Box 546545
Cambridge, MA 02142
USA

X/Open - (UK)
X/Open Company Limited
Abbots House
Abbey Street
Reading
Berkshire, RG1 3BD,
United Kingdom

X/Open - (USA)
X/Open Company Limited
1010 El Camino Real, Suite 380
Menlo Park, CA 94025
USA

International ANSI Distributors:

American Technical Publishers, Ltd.
27/29 Knowl Piece, Wilbury Way
Hertfordshire, SG4 OSX,
United Kingdom

Japanese Standards Association
1-24, Akasaka
4-Chome, Minato-ku,
Tokyo 107
Japan

Standards Council of Canada
45 O'Connor Street, Suite 1200
Ottawa K1P 6N7, Ontario
Canada

ISO Member Bodies (ISO Sources):

American National Standards Institute
11 West 42nd Street
13th Floor
New York, NY 10036
USA

DIN Deutsches Institut für Normung
Burggrafenstrasse 6
Postfach 1107
D-1000 Berlin 30
Germany/Allemagne (DIN)

JSA (Japan Standards Association)
1-24 Akasaka
4-Chome, Minato-ku
Tokyo 107
Japan

Standards Council of Canada
45 O'Connor Street, Suite 1200
Ottawa K1P 6N7, Ontario
Canada

E

Entry Points

This section lists, in alphabetical order, the required entry points as specified by Layers 1 and 2 of this API. In addition, the conformance to industry or defacto standards of the entry points has been indicated. Entry points for options are not included in these tables.

The first table lists entry points that are required by the PRO ABI and are primarily for system platform implementations, not typical application APIs. These entry points are in addition to those specified by the standards in Layers 1 and 2. The second table lists the system calls and library functions which are directly available to application programs. And, finally, a third table lists the user commands and utilities available to interactive users, application scripts, and to application programs through the "system()" call.

How to read the tables:

The dagger symbol (†) indicates that this entry will be withdrawn in future editions of XPG, therefore, PRO does not recommend using these entry points.

The double-dagger symbol (‡) indicates that this entry will be mandatory in future editions of XPG.

The star symbol (*) indicates that this entry is a data attribute, not a function.

The diamond symbol (◆) indicates that this entry is part of the Encryption feature group of XPG4. The interfaces to these entry points must exist; however, because of export restrictions imposed by the U.S. Goverment with regard to the decoding algorithm, implementations are restricted in making these functions available. See X/Open System Interfaces and Headers, Issue 4 for further information.

The columns indicating standards conformance are grouped according to the layer in which they are specified; the groupings are labelled with headings at the top of the matrix as Layer 1 and Layer 2 respectively. The standards that com-

prise each layer are stated in order of precedence from highest on the left (POSIX 1003.1) to lowest on the right (AES).

For ease of layer identification and readability, the rows that conform to Layer 1 are shaded.

ABI Required Entry Points

			LAYER 1			LAYER 2					
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 2	XPG 3	XPG 4	AES	HP
_filbuf	libc	getc macro									■
_flsbuf	libc	putc macro									■
ioctl()	libc	control device					■				
ptrace()	libc	process trace						■			
ptsname()	libc	get the name of a slave pty									■
sbrk()	libc	change data segment space allocation					■				
shl_definesym()	libdld	adds a symbol to the shared library symbol table for the current process making it the most visible definition.									■
shl_findsym	libdld	obtains the address of an exported symbol sym from a shared library.									■
shl_get	libdld	returns information about currently loaded libraries, including those loaded implicitly at startup time.									■
shl_gethandle	libdld	returns information about the library specified by the handle argument.									■

ABI Required Entry Points

			LAYER 1			LAYER 2					
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 2	XPG 3	XPG 4	AES	HP
shl_getsymbols	libdld	provides an array of symbol records, allocated using the supplied memory allocator, that are associated with the library specified by handle.									■
shl_load	libdld	attaches the shared library named by path to the process.									■
shl_unload	libdld	can be used to detach a shared library from the process.									■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
abort()	libc	generate an abnormal process abort			■		■	■
abs()	libc	return integer absolute value	■		■		■	■
access()	libc	determine accessibility of a file	■				■	
acos()	libM	arc cosine function	■		■		■	■
alarm()	libc	schedule an alarm signal	■				■	■
asctime()	libc	convert date and time to string	■		■		■	■
asin()	libM	arc sine function	■		■		■	■
assert()		insert program diagnostics	■		■		■	■
atan()	libM	arc tangent function	■		■		■	■
atan2()	libM	arc tangent function	■		■		■	■
atexit()	libc	register function to run at process termination			■		■	
atof()	libc	convert string to double-precision number	■		■		■	■
atoi()	libc	convert string to integer	■		■		■	■
atol()	libc	convert string to long integer	■		■		■	■
bsearch()	libc	binary search a sorted table	■		■		■	■
calloc()	libc	memory allocator	■		■		■	■
catclose()	libc	close a message catalog descriptor					■	■
catgets()	libc	read a program message					■	■
catopen()	libc	open a message catalog descriptor					■	■
ceil()	libM	ceiling value function	■		■		■	■
cfgetispeed()	libc	get input baud rate	■				■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
cfgetospeed()	libc	get output baud rate	■				■	■
cfsetispeed()	libc	set input baud rate	■				■	■
cfsetospeed()	libc	set output baud rate	■				■	■
chdir()	libc	change working directory	■				■	■
chmod()	libc	change access mode of file	■				■	■
chown()	libc	change owner and group of a file	■				■	■
† chroot()	libc	change root directory					■	■
clearenv()	libc	clear the process environment						■
clearerr()	libc	clear indicators on a stream	■		■		■	■
clock()	libc	report CPU time used			■		■	■
close()	libc	close a file descriptor	■				■	■
closedir()	libc	close a directory stream	■				■	■
confstr()	libc	get configurable variables				■	■	
cos()	libM	cosine function	■		■		■	■
cosh()	libM	hyperbolic cosine function	■		■		■	■
creat()	libc	create a new file or rewrite an existing one	■				■	■
◆ crypt()		string encoding function					■	
ctermid()	libc	generate pathname for controlling terminal	■	■			■	■
ctime()	libc	convert time value to date and time string	■		■		■	■
† cuserid()	libc	character login name of the user	■				■	■
* daylight	libc	daylight savings time flag					■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
difftime()	libc	compute the difference between two calendar time values			■		■	■
div()	libc	compute quotient and remainder of an integer division			■		■	■
drand48()	libc	generate uniformly distributed pseudo-random numbers					■	■
dup()	libc	duplicate an open file descriptor	■				■	■
dup2()	libc	duplicate an open file descriptor	■				■	■
◆ encrypt()		encoding function					■	
* environ	libc	array of character pointers to the environment strings	■				■	■
erand48()	libc	generate uniformly distributed pseudo-random numbers					■	■
erf()	libM	error function					■	■
erfc()	libM	complementary error function					■	■
* errno	libc	error indicator for function calls	■		■		■	■
execl	libc	execute a file	■				■	■
execle	libc	execute a file	■				■	■
execlp	libc	execute a file	■				■	■
execv	libc	execute a file	■				■	■
execve	libc	execute a file	■				■	■
execvp	libc	execute a file	■				■	■
_exit()	libc	terminate process	■				■	■
exit()	libc	terminate process	■		■		■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
exp()	libM	exponential function	■		■		■	■
fabs()	libM	absolute value function	■		■		■	■
fchmod()	libc	change a file's access permissions and attributes						■
fchown()	libc	change a file's owner and/ or group ID						■
fclose()	libc	close a stream	■		■		■	■
fcntl()	libc	file control	■				■	■
fdopen()	libc	associate a stream with a file descriptor	■				■	■
feof()	libc	test end-of-file indicator on a stream	■		■		■	■
ferror()	libc	test error indicator on a stream	■		■		■	■
fflush()	libc	flush a stream	■		■		■	■
fgetc()	libc	get a byte from a stream	■		■		■	■
fgetpos()	libc	get current file position information			■		■	■
fgets()	libc	get a string from a stream	■		■		■	■
fgetwc()	libc	get a wide-character code from a stream					■	
fgetws()	libc	get a wide character string from a stream file					■	
fileno()	libc	map stream pointer to file descriptor	■				■	■
floor()	libM	floor function	■		■		■	■
fmod()	libM	floating-point remainder value function	■		■		■	■
fnmatch()	libc	match filename or path-name				■	■	
fopen()	libc	open a stream	■		■		■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
fork()	libc	create a new process	■				■	■
fpathconf()	libc	get configurable path-name variables	■			■	■	■
fprintf()	libc	print formatted output	■		■		■	■
fputc()	libc	put byte on a stream	■		■		■	■
fputs()	libc	put a string on a stream	■		■		■	■
fputwc()	libc	put wide-character code on a stream					■	
fputws()	libc	put a wide character string on a stream file					■	
fread()	libc	binary input	■		■		■	■
free()	libc	free allocated memory	■		■		■	■
freopen()	libc	open a stream	■		■		■	■
frexp()	libc	extract mantissa and exponent from double-precision number	■		■		■	■
fscanf()	libc	convert formatted input	■		■		■	■
fseek()	libc	reposition a file-position indicator in a stream	■		■		■	■
fsetpos()	libc	set current file position			■		■	■
fstat()	libc	get file status	■				■	■
fsync()	libc	synchronize a file's in-core state with its state on disk					■	■
ftell()	libc	return a file offset in a stream	■		■		■	■
ftruncate()	libc	change file length						■
ftw()	libc	walk a file tree					■	■
fwrite()	libc	binary output	■		■		■	■
† gamma()	libM	log gamma function					■	■
getc()	libc	get byte from a stream	■		■		■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
getchar()	libc	get byte from stdin stream	■		■		■	■
getclock()	libc	get current value of system-wide clock						■
getcwd()	libc	get pathname of current working directory	■				■	■
getegid()	libc	get effective group ID	■				■	■
getenv()	libc	return value for environment name	■		■		■	■
geteuid()	libc	get effective user ID	■				■	■
getgid()	libc	get real group ID	■				■	■
getgrgid()	libc	get group database entry for particular group ID	■				■	■
getgrnam()	libc	search group database for particular name	■				■	■
getgroups()	libc	get supplementary group IDs	■				■	■
getlogin()	libc	get login name	■				■	■
getopt()	libc	command option parsing				■	■	■
† getpass()	libc	read a password					■	■
getpgrp()	libc	get process group ID	■				■	■
getpid()	libc	get process ID	■				■	■
getppid()	libc	get parent process ID	■				■	■
getpwnam()	libc	search user database for particular name	■				■	■
getpwuid()	libc	search user database for particular user ID	■				■	■
gets()	libc	get a string from stdin stream	■		■		■	■
gettimer()	libc	get value of a per-process timer						■
getuid()	libc	get real user ID	■				■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
getw()	libc	get a word from a stream					■	■
getwc()	libc	get a wide character from a stream					■	
getwchar()	libc	get a wide character from stdin stream					■	
glob()	libc	generate pathnames matching a pattern				■	■	
globfree()	libc	generate pathnames matching a pattern				■	■	
gmtime()	libc	convert time value to broken-down UTC time	■		■		■	■
hcreate()	libc	manage hash search tables					■	■
hdestroy()	libc	manage hash search tables					■	■
hsearch()	libc	manage hash search tables					■	■
hypot()	libM	Euclidean distance function, complex absolute value					■	■
iconv()	libc	code conversion function					■	
iconv_close()	libc	code conversion deallocation function					■	
iconv_open()	libc	code conversion deallocation function					■	
isalnum()	libc	test for alphanumeric character	■		■		■	■
isalpha()	libc	test for alphabetic character	■		■		■	■
isascii()	libc	test for 7-bit US-ASCII character					■	■
isatty()	libc	test for a terminal device	■				■	■
iscntrl()	libc	test for control character	■		■		■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
isdigit()	libc	test for decimal digit	■		■		■	■
isgraph()	libc	test for visible character	■		■		■	■
islower()	libc	test for lower-case letter	■		■		■	■
isnan()	libM	test for NaN functions					■	■
isprint()	libc	test for printing character	■		■		■	■
ispunct()	libc	test for punctuation character	■		■		■	■
isspace()	libc	test for white-space character	■		■		■	■
isupper()	libc	test for upper-case letter	■		■		■	■
iswalnum()	libc	test for an alphanumeric wide-character code					■	
iswalpha()	libc	test for an alphabetic wide-character code					■	
iswcntrl()	libc	test for a control wide-character code					■	
iswctype()	libc	test character for specified class					■	
iswdigit()	libc	test for a decimal digit wide-character code					■	
iswgraph()	libc	test for a visible wide-character code					■	
iswlower()	libc	test for a lower-case letter wide-character code					■	
iswprint()	libc	test for a printing wide-character code					■	
iswpunct()	libc	test for a punctuation wide-character code					■	
iswspace()	libc	test for a white-space wide-character code					■	
iswupper()	libc	test for an upper-case letter wide-character code					■	

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
iswxdigit()	libc	test for a hexadecimal digit wide-character code					■	
isxdigit()	libc	test for hexadecimal digit	■		■		■	■
j0()	libM	Bessel function of the first kind					■	■
j1()	libM	Bessel function of the first kind					■	■
jn()	libM	Bessel function of the first kind					■	■
jrand48()	libc	generate uniformly distributed pseudo-random numbers					■	■
kill()	libc	send a signal to a process or a group of processes	■				■	■
labs()	libc	return long integer absolute value			■		■	■
lcong48()	libc	generate uniformly distributed pseudo-random numbers					■	■
ldexp()	libc	load exponent of a floating point number	■		■		■	■
ldiv()	libc	compute quotient and remainder of a long division			■		■	■
lfind()	libc	find entry in linear search table					■	■
lgamma()	libM	log gamma function					■	■
link()	libc	link to a file	■				■	■
localeconv()	libc	query the numeric formatting conventions of the current locale			■		■	■
localtime()	libc	convert time value to broken-down local time	■		■		■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
log()	libM	natural logarithm function	■		■		■	■
log10()	libM	base 10 logarithm function	■		■		■	■
longjmp()	libc	non-local goto	■		■		■	■
lrand48()	libc	generate uniformly distributed pseudo-random numbers					■	■
lsearch()	libc	linear search and update					■	■
lseek()	libc	move read/write file pointer; seek	■				■	■
lstat()	libc	get file status						■
madvise()	libc	advise the system of a process' expected paging behavior						■
malloc()	libc	memory allocator	■		■		■	■
mblen()	libc	get number of bytes in a character			■		■	■
mbstowcs()	libc	convert a character string to a wide character string			■		■	■
mbtowc()	libc	convert a character to a wide character code			■		■	■
memccpy()	libc	copy bytes in memory					■	■
memchr()	libc	find byte in memory			■		■	■
memcmp()	libc	compare bytes in memory			■		■	■
memcpy()	libc	copy bytes in memory			■		■	■
memmove()	libc	copy bytes in memory with overlapping areas			■		■	■
memset()	libc	set bytes in memory			■		■	■
mkdir()	libc	make a directory file	■				■	■
mkfifo()	libc	make a FIFO file	■				■	■
mktime()	libc	convert broken-down time into time since the Epoch	■		■		■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
mktimer()	libc	allocate a per-process timer						■
mmap()	libc	map file system object into virtual memory						■
modf()	libc	decompose floating point number	■		■		■	■
mprotect()	libc	modify access protections of memory mapping						■
mrand48()	libc	generate uniformly distributed pseudo-random numbers					■	■
msem_init()	libc	initialize a semaphore in a mapped file or shared memory region						■
msem_lock()	libc	lock a semaphore						■
msem_remove()	libc	remove a semaphore						■
msem_unlock()	libc	unlock a semaphore						■
msgctl()	libc	message control operations					■	
msgget()	libc	get message queue					■	
msgrcv()	libc	message receive operation					■	
msgsnd()	libc	message send operation					■	
msync()	libc	synchronize a mapped file						■
munmap()	libc	unmap a mapped region						■
nice()	libc	change priority of a process					■	■
nl_langinfo()	libc	language information					■	■
nrand48()	libc	generate uniformly distributed pseudo-random numbers					■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
open()	libc	open file for reading or writing	■				■	■
opendir()	libc	open directory	■				■	■
* optarg	libc	command option parsing				■	■	■
* opterr	libc	command option parsing				■	■	■
* optind	libc	command option parsing				■	■	■
* optopt	libc	command option parsing					■	
pathconf()	libc	get configurable path-name variables	■			■	■	■
pause()	libc	suspend process until signal	■				■	■
pclose()	libc	close a pipe stream to or from a process				■	■	■
perror()	libc	write error messages to standard error	■		■		■	■
pipe()	libc	create an interprocess channel	■				■	■
poll()	libc	monitor I/O conditions on multiple file descriptors						■
popen()	libc	initiate pipe streams to or from a process				■	■	■
pow()	libM	power function	■		■		■	■
printf()	libc	print formatted output	■		■		■	■
putc()	libc	put byte on a stream	■		■		■	■
putchar()	libc	put byte on a stdout stream	■		■		■	■
putenv()	libc	change or add value to environment					■	■
puts()	libc	put a string on standard output	■		■		■	■
putw()	libc	put a word on a stream					■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
putwc()	libc	put wide character on a stream					■	
putwchar()	libc	put wide character on std-out stream					■	
qsort()	libc	quicker sort	■		■		■	■
raise()	libc	send a signal to a process or a group of processes			■		■	■
rand()	libc	pseudo-random number generator	■		■		■	■
read()	libc	read input	■				■	■
readdir()	libc	read directory	■				■	■
readlink()	libc	reads the value of a symbolic link						■
realloc()	libc	memory reallocator	■		■		■	■
regcomp()	libc	regular expression matching				■	■	
regerror()	libc	regular expression matching				■	■	
regexec()	libc	regular expression matching				■	■	
regfree()	libc	regular expression matching				■	■	
† regexp():		regular expression compile and match routines						
† compile()							■	■
† step()							■	■
† advance()							■	■
*† loc1							■	■
*† loc2							■	■
*† locs							■	■
reltimer()	libc	relatively arm a per-process timer						■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
remove()	libc	remove a file	■		■		■	■
rename()	libc	change the name of a file	■		■		■	■
rewind()	libc	reset file position indicator in a stream	■		■		■	■
rewinddir()	libc	reset position of directory stream to the beginning of a directory	■				■	■
rmdir()	libc	remove a directory file	■				■	■
rmtimer()	libc	free a per-process timer						■
scanf()	libc	convert formatted input	■		■		■	■
seed48()	libc	generate uniformly distributed pseudo-random numbers					■	■
seekdir()	libc	set position of directory stream					■	■
semctl()	libc	semaphore control operations					■	
semget()	libc	get set of semaphores					■	
semop()	libc	semaphore operations					■	
setbuf()	libc	assign buffering to a stream	■		■		■	■
setclock()	libc	set value of system-wide clock						■
setgid()	libc	set group ID	■				■	■
setgroups()	libc	set the group access list						■
setjmp()	libc	set jump point for a non-local goto	■		■		■	■
♦ setkey()		set encoding key					■	
setlocale()	libc	set and get the locale of a program	■		■		■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
setpgid()	libc	set process group ID for job control	■				■	■
setsid()	libc	create session and set process group ID	■				■	■
setuid()	libc	set user ID	■				■	■
setvbuf()	libc	assign buffering to a stream			■		■	■
shmat()	libc	shared memory attach operation					■	
shmctl()	libc	shared memory control operations					■	
shmdt()	libc	shared memory detach operation					■	
shmget()	libc	get shared memory segment					■	
sigaction()	libc	examine and change signal action	■				■	■
sigaddset()	libc	add a signal to a signal set	■				■	■
sigdelset()	libc	delete a signal from a signal set	■				■	■
sigemptyset()	libc	initialize and empty a signal set	■				■	■
sigfillset()	libc	initialize and fill a signal set	■				■	■
sigismember()	libc	test for a signal in a signal set	■				■	■
siglongjmp()	libc	non-local goto with signal handling	■				■	■
signal()	libc	specify what to do upon receipt of a signal			■		■	■
†* signgam	libM	log gamma function					■	■
sigpending()	libc	examine pending signals	■				■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
sigprocmask()	libc	examine and change blocked signals	■				■	■
sigsetjmp()	libc	set jump point for a non-local goto	■				■	■
sigsuspend()	libc	wait for a signal					■	
sin()	libM	sine function	■		■		■	■
sinh()	libM	hyperbolic sine function	■		■		■	■
sleep()	libc	suspend execution for interval	■				■	■
sprintf()	libc	print formatted output	■		■		■	■
sqrt()	libM	square root function	■		■		■	■
srand()	libc	seed simple pseudo-random number generator	■		■		■	■
srand48()	libc	generate uniformly distributed pseudo-random numbers					■	■
sscanf()	libc	convert formatted input	■		■		■	■
stat()	libc	get file status	■				■	■
stdio():	libc	standard buffered input/output stream file package						
* stderr	libc		■		■		■	■
* stdin	libc		■		■		■	■
* stdout	libc		■		■		■	■
strcat()	libc	concatenate two strings	■		■		■	■
strchr()	libc	string scanning operation	■		■		■	■
strcmp()	libc	compare two strings	■		■		■	■
strcoll()	libc	string comparison using collating information			■		■	■
strcpy()	libc	copy a string	■		■		■	■
strcspn()	libc	geet length of complementary string	■		■		■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
strerror()	libc	get error message string			■		■	■
‡ strfmon	libc	convert monetary value to string					■	
strftime()	libc	convert date and time to string	■		■		■	■
strlen()	libc	get string length	■		■		■	■
strncat()	libc	concatenate part of two strings	■		■		■	■
strncmp()	libc	compare part of two strings	■		■		■	■
strncpy()	libc	copy part of a string	■		■		■	■
strpbrk()	libc	scan string for byte	■		■		■	■
‡ strptime()		date and time conversion					■	
strrchr()	libc	string scanning operation	■		■		■	■
strspn()	libc	get length of substring	■		■		■	■
strstr()	libc	find substring	■		■		■	■
strtod()	libc	convert string to double-precision number			■		■	■
strtok()	libc	split string into tokens	■		■		■	■
strtol()	libc	convert string to long integer			■		■	■
strtoul()	libc	convert string to unsigned long			■		■	■
strxfrm()	libc	string transformation			■		■	■
swab()	libc	swap bytes					■	■
symlink()	libc	make symbolic link						■
sysconf()	libc	get configurable system variables	■			■	■	■
system()	libc	issue a shell command			■	■	■	■
tan()	libM	tangent function	■		■		■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
tanh()	libM	hyperbolic tangent function	■		■		■	■
tcdrain()	libc	wait for transmission of output	■				■	■
tcflow()	libc	suspend or restart the transmission or reception of data	■				■	■
tcflush()	libc	flush non-transmitted output data, non-read input data or both	■				■	■
tcgetattr()	libc	get the parameters associated with the terminal	■				■	■
tcgetpgrp()	libc	get foreground process group ID	■				■	■
tcsendbreak()	libc	send a "break" for a specific duration	■				■	■
tcsetattr()	libc	set the parameters associated with the terminal	■				■	■
tcsetpgrp()	libc	set foreground process group id	■				■	■
tdelete()	libc	delete node from binary search tree					■	■
telldir()	libc	current location of a named directory stream					■	■
tempnam()	libc	create a name for a temporary file					■	■
tfind()	libc	search binary search tree					■	■
time()	libc	get time	■		■		■	■
times()	libc	get process and child process times	■				■	■
* timezone	libc	difference from UTC and local standard time					■	■
tmpfile()	libc	create a temporary file	■		■		■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
tmpnam()	libc	create a name for a temporary file	■		■		■	■
toascii()	libc	translate integer to a 7-bit ASCII character					■	■
_tolower()	libc	transliterate upper-case characters to lower-case					■	■
tolower()	libc	transliterate upper-case characters to lower-case	■		■		■	■
_toupper()	libc	transliterate lower-case characters to upper-case					■	■
toupper()	libc	transliterate lower-case characters to upper-case	■		■		■	■
towlower()	libc	transliterate upper-case wide-character code to lower-case					■	
towupper()	libc	transliterate lower-case wide-character code to upper-case					■	
truncate()	libc	change file length						■
tsearch()	libc	manage binary search tree					■	■
ttyname()	libc	find pathname of a terminal	■				■	■
twalk()	libc	traverse binary search tree					■	■
* tzname[ ]	libc	timezone strings	■				■	■
tzset()	libc	set time zone conversion information	■				■	■
ulimit()	libc	get and set user limits					■	■
umask()	libc	set and get file creation mask	■				■	■
uname()	libc	get/set name of current HP-UX system	■				■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
ungetc()	libc	push character back into input stream	■		■		■	■
ungetwc()	libc	push a wide character back into an input stream					■	
unlink()	libc	remove directory entry; delete file	■				■	■
utime()	libc	set file access and modification times	■				■	■
vfprintf()	libc	format output of a stdarg argument list			■		■	■
vprintf()	libc	format output of a stdarg argument list			■		■	■
vsprintf()	libc	format output of a stdarg argument list			■		■	■
wait()	libc	wait for child process to stop or terminate	■				■	■
waitpid()	libc	wait for child process to stop or terminate	■				■	■
wcscat()	libc	concatenate two wide character strings					■	
wcschr()	libc	wide character string scanning operation					■	
wcscmp()	libc	compare two wide character strings					■	
‡ wcscoll()	libc	wide character string comparison using collating information					■	
wcscpy()	libc	copy a wide character string					■	
wcscspn()	libc	get length of complementary wide substring					■	
‡ wcsftime()	libc	convert date and time to wide-character string					■	

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
wcslen()	libc	get wide character string length					■	
wcsncat()	libc	concatenate part of two wide character strings					■	
wcsncmp()	libc	compare part of two wide character strings					■	
wcsncpy()	libc	copy part of a wide character string					■	
wcspbrk()	libc	scan wide character string for wide-character code					■	
wcsrchr()	libc	wide character string scanning operation					■	
wcsspn()	libc	get length of wide substring					■	
wcstod()	libc	convert wide character string to double-precision number					■	
wcstok()	libc	split wide character string into tokens					■	
wcstol()	libc	convert wide character string to long integer					■	
wcstombs()	libc	convert a wide character string to a character string			■		■	■
wcstoul()	libc	convert wide character string to unsigned long					■	
wcswcs()	libc	find wide substring					■	
wcswidth()	libc	number of column positions of a wide character string					■	
‡ wcsxfrm()	libc	wide character string transformation					■	
wctomb()	libc	convert a wide character code to a character			■		■	■

System Calls and Library Functions

			LAYER 1			LAYER 2		
Command	Library	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
wctype()	libc	define character class					■	
wcwidth()	libc	number of column positions of a wide-character code					■	
wordexp()		perform word expansions				■	■	
wordfree()		perform word expansions				■	■	
write()	libc	write on a file	■				■	■
y0()	libM	Bessel function of the second kind					■	■
y1()	libM	Bessel function of the second kind					■	■
yn()	libM	Bessel function of the second kind					■	■

User Commands and Utilities

		LAYER 1			LAYER 2		
Command	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
admin	create and administer SCCS files					■	
alias	define or display aliases				■	■	
ar	maintain portable archives and libraries				■	■	
asa	interpret ASA carriage control characters				■	■	
at	execute commands at a later time				■	■	
awk	pattern-directed scanning and processing language				■	■	
basename	return non-directory portion of pathname				■	■	
batch	execute commands when the system load permits				■	■	
bc	arbitrary-precision arithmetic language				■	■	
bg	run jobs in the background				■	■	
c89	C compiler - POSIX compliant				■	■	
cal	print calendar					■	
† calendar	reminder service					■	
cancel	cancel line printer requests					■	
cat	concatenate, copy, and print files				■	■	
† cc	C compiler					■	
cd	change working directory				■	■	
cflow	generate C flow graph					■	
chgrp	change file group ownership				■	■	
chmod	change file mode				■	■	
chown	change file ownership				■	■	
cksum	print file checksum and sizes				■	■	
cmp	compare two files				■	■	

User Commands and Utilities

		LAYER 1			LAYER 2		
Command	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
† col	filter reverse line-feeds and back-spaces					■	
comm	select or reject lines common to two sorted files				■	■	
command	execute a simple command				■	■	
compress	comptess data					■	
cp	copy files and directory subtrees				■	■	
† cpio	copy file archives in and out					■	
cpp	the C language preprocessor						
crontab	scedule periodic background work				■	■	
csplit	split files based on context				■	■	
ctags	create a tags file				■	■	
cu	call another (UNIX) system; terminal emulator					■	
cut	cut out (extract) selected fields of each line of a file				■	■	
cxref	generate C program cross-reference table					■	
date	print or set the date and time				■	■	
dd	convert, reblock, translate, and copy a (tape) file				■	■	
delta	make a delta (change) to an SCCS file					■	
df	report number of free disk blocks				■	■	
diff	differential file and directory comparator				■	■	
† dircmp	directory comparison					■	
dirname	return directory portion of pathname				■	■	
du	summarize disk usage				■	■	
echo	echo (print) arguments				■	■	

User Commands and Utilities

		LAYER 1			LAYER 2		
Command	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
ed	text editor				■	■	
egrep	search a file with an ERE (extended regular expression) pattern				■	■	
env	set environment for command execution				■	■	
ex	extended line-oriented text editor				■	■	
expand	convert tabs to spaces				■	■	
expr	evaluate arguments as an expression				■	■	
false	return false value				■	■	
fc	process command history list					■	
fg	run jobs in the foreground				■	■	
fgrep	search a file for a fixed-string pattern				■	■	
file	determine file type				■	■	
find	find files				■	■	
fold	fold long lines for finite width output device				■	■	
fort77	FORTRAN 77 compiler				■	■	
gencat	generate a formatted message catalog file					■	
get	get a version of an SCCS file					■	
getconf	get system configuration values				■	■	
getopts	parse utility (command) options				■	■	
grep	search a file for a pattern				■	■	
hash	remember or report utility locations					■	
head	give first few lines				■	■	
iconv	codeset conversion					■	
id	return user identity				■	■	

User Commands and Utilities

		LAYER 1			LAYER 2		
Command	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
jobs	display status of jobs in the current session				■	■	
join	relational database operator				■	■	
kill	terminate a process				■	■	
lex	generate programs for lexical analysis of text				■	■	
† line	read one line from user input					■	
† lint	a C program checker/verifier					■	
ln	link files and directories				■	■	
locale	get locale-specific (NLS) information				■	■	
localedef	define locale environment				■	■	
logger	make entries in the system log				■	■	
logname	get login name				■		
lp	send requests to an LP line printer or plotter				■	■	
lpstat	print LP status information					■	
ls	list contents of directories				■	■	
m4	macro processor					■	
† mail	send mail to users					■	
mailx	interactive message processing system				■	■	
make	maintain, update, and regenerate groups of programs				■	■	
man	display system documentation				■	■	
mesg	permit or deny messages to terminal				■	■	
mkdir	make a directory				■	■	
mkfifo	make FIFO (named pipe) special files				■	■	

User Commands and Utilities

		LAYER 1			LAYER 2		
Command	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
more	display files on a page-by-page basis				■	■	
mv	move or rename files and directories				■	■	
newgrp	log in to a new group				■	■	
nice	invoke a utility with an altered system scheduling priority				■	■	
nl	line numbering filter					■	
nm	print name list of common object file.				■	■	
nohup	run a command immune to hangups, logouts, and quits				■	■	
od	dump files in various formats					■	
† pack	compress files					■	
paste	merge same lines of several files or subsequent lines of one file				■	■	
patch	apply changes to files				■	■	
pathchk	check pathnames				■	■	
pax	portable archive exchange				■	■	
† pcat	expand and concatenate files					■	
† pg	file perusal filter for soft-copy terminals					■	
pr	print files				■	■	
printf	format and print arguments				■	■	
prs	print and summarize an SCCS file					■	
ps	report process status				■	■	
pwd	working directory name				■	■	
read	read a line from standard input				■	■	
renice	set system scheduling priorities of running processes				■	■	

User Commands and Utilities

		LAYER 1			LAYER 2		
Command	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
rm	remove files or directories				■	■	
rmdel	remove a delta from an SCCS file					■	
rmdir	remove directories				■	■	
sact	print current SCCS file editing activity					■	
sccs	front end for the SCCS subsystem					■	
sed	stream text editor				■	■	
sh	shell, the standard command language interpreter					■	
sleep	suspend execution for an interval				■	■	
sort	sort or merge files				■	■	
† spell	find spelling errors					■	
split	split a file into pieces				■	■	
strings	find printable strings in files				■	■	
strip	strip symbol and line number information from an object file				■	■	
stty	set the options for a terminal port				■	■	
† sum	print checksum and block or byte count of file(s)					■	
tabs	set tabs on a terminal				■	■	
tail	deliver the last part of a file				■	■	
talk	talk to another user				■	■	
† tar	tape file archiver					■	
tee	pipe fitting				■	■	
test	evaluate expression				■	■	
time	time a command				■	■	
touch	update access, modification, and/or change times of file				■	■	
tput	change terminal characteristics				■	■	

User Commands and Utilities

		LAYER 1			LAYER 2		
Command	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
tr	translate characters				■	■	
true	return true value				■	■	
tsort	topological sort					■	
tty	get the name of the terminal				■	■	
type	write a description of command type					■	
ulimit	set or report file size limit					■	
umask	get or set the file mode creation mask				■	■	
unalias	remove alias definitions				■	■	
uname	return system name				■	■	
uncompress	expand a compressed file					■	
unexpand	convert spaces to tabs				■	■	
unget	undo a previous get of an SCCS file					■	
uniq	report repeated lines in a file				■	■	
† unpack	expand files					■	
uucp	UNIX system to UNIX system copy					■	
uudecode	decode a binary file				■	■	
uuencode	encode a binary file				■	■	
uulog	query system-to-system transaction log					■	
uuname	list names of other known uucp systems					■	
uupick	receive public system-to-system file copies					■	
uustat	uucp status inquiry and job control					■	
uuto	public UNIX system to UNIX system file copy					■	

User Commands and Utilities

		LAYER 1			LAYER 2		
Command	Description	POSIX 1003.1	FIPS 151-1	ANSI C	POSIX 1003.2	XPG 4	AES
uux	UNIX system to UNIX system command execution					■	
val	validate SCCS file					■	
vi	screen-oriented (visual) disply editor				■	■	
wait	await process completion				■	■	
wc	word, line, and character count				■	■	
what	get SCCS identification information					■	
who	who is on the system				■	■	
write	interactively write (talk) to another user				■	■	
xargs	construct argument list(s) and execute command				■	■	
yacc	yet another compiler-compiler				■	■	
zcat	expand and concatenate data					■	

F

Entry Point Supplement

The following entry point documentation is supplemental to the relevant standard specifications. These supplemental specifications are intended to:

Resolve ambiguities which are not resolved by the standards precedence model.

Define extensions to the standard definitions that are required for PRO Conformance.

Define implementation-dependent attributes that are common to (and required of) all PRO Conformant systems.

The REMARKS sections describe the nature of the supplemental specification, and whether the supplement is a *Clarification*, an *Extension*, or a PRO *Attribute*. The DESCRIPTION section provides additional detail of the entry's function and behavior.

Note: These entry points are subject to withdrawal in future versions of this document.

NAME

_filbuf( ) - getc macro

SYNOPSIS

_filbuf(FILE *stream)

REMARKS

Fills an internal buffer and returns the first character of that buffer.
*stream.__cnt is set to the number of characters remaining in the buffer, while
*stream.__ptr is set to point to the next character in the buffer.

NAME

_flsbuf( ) - putc macro

SYNOPSIS

int _flsbuf(unsigned char c, FILE *stream)

REMARKS

Appends c to an internal buffer of size n, and writes the buffer to stream. The value of *stream.__ptr is reset to the top of the buffer and *stream.__cnt is set to (n-1).

NAME

cuserid( ) - get character login name of the user

SYNOPSIS

```
#include <unistd.h>
#include <stdio.h>
char *cuserid(char *s);
```

REMARKS

Clarification: PRO conformant implementations of the **cuserid**( ) function will return the effective user id of the process.

Because this function behaves differently across many different operating system implementations, its use is not recommended. It is provided only for conformance to current industry standards. **cuserid**( ) has been removed from POSIX 1003.1, is marked for withdrawal in XPG4, and will be withdrawn from PRO standards in a future revision of the PRO API. For portability and security, application writers should use one of the following calls, depending on which user name is desired:

getpwuid(geteuid( ))	*Equivalent to POSIX 1003.1, HP-UX, & PRO cuserid( )*
getpwuid(getuid( ))	*Historical implementation of cuserid( )*
getlogin( )	*Return user's login name.*

NAME

ioctl( ) - generic device control commands

SYNOPSIS

#include <sys/ioctl.h>

int ioctl (int *fildes*, int *request*, ... /* *arg* */);

REMARKS:

Extension: **ioctl( )** is provided to support BSD networking.

DESCRIPTION

The **ioctl( )** system call provides for control over open devices. This include file describes requests and arguments used in **ioctl( )** which are of a generic nature. How individual requests will affect any particular device are implementation specific. If a device does not support an ioctl request it returns EINVAL.

fildes an open file descriptor

request selects the control function to be performed and will depend on the device being addressed.

arg represents additional information that is needed by this specific device to perform the requested function. The data type of *arg* depends upon the particular control request, but it is either an integer or a pointer to a device-specific data structure.

RETURN VALUE

If an error has occurred, a value of -1 is returned and **errno** is set to indicate the error.

ioctl( ) fails if one or more of the following are true:

[EBADF] *fildes* is not a valid open file descriptor.

[ENOTTY] The request is not appropriate to the selected device.

[EINVAL] *request* or *arg* is not valid.

[EINTR] A signal was caught during the **ioctl( )** system call.

NAME

ln - link files and directories

SYNOPSIS

ln [-f] [-s] *file1 new_file*

ln [-f] [-s] *file1 [file2 ...] dest_directory*

ln [-f] [-s] *directory1 [directory2 ...] dest_directory*

REMARKS

PRO conformant implementations of the ln command will support the **-s** option, in addition to the **-f** option specified by XPG4 and POSIX 1003.2.

DESCRIPTION

Extension: The **-s** option creates symbolic links instead of the usual hard links. A symbolic link contains the name of the file to which it is linked. The referenced file is used when an **open( )** operation is performed on the link. A **stat( )** on a symbolic link returns the linked-to file; an **lstat( )** must be performed to obtain information about the link. The **readlink( )** call can be used to read the contents of the symbolic link. Symbolic links can span file systems and can refer to directories.

NAME

mmap( ) - map object into virtual memory

SYNOPSIS

```
#include <sys/mman.h>
    caddr_t mmap(
    caddr_t addr,
    size_t len,
    int prot,
    int flags,
    int fildes,
    off_t off);
```

REMARKS

Clarification: Support of **MAPPED_FIXED** is not required on PRO conformant systems.

Because the PA-RISC memory architecture utilizes a globally shared virtual address space between processes, and discourages multiple virtual address translations to the same physical address, all concurrently existing **MAP_SHARED** mappings of a file range must share the same virtual address offsets and hardware translations. PRO compliant systems allocate virtual address ranges for shared memory and shared mapped files in the range 0x80000000 through 0xefffffff. This address range is used globally for all memory objects shared between processes.

This implies the following:

- Any single range of a file cannot be mapped multiply into different virtual address ranges.
- After the initial **MAP_SHARED mmap( )** of a file range, all subsequent **MAP_SHARED** calls to **mmap( )** to map the same range of a file must either specify **MAP_VARIABLE** in flags and inherit the virtual address range the system has chosen for this range, or specify **MAP_FIXED** with an addr that corresponds exactly to the address chosen by the system for

the initial mapping. Only after all mappings for a file range have been destroyed can that range be mapped to a different virtual address.

- In most cases, two separate calls to **mmap( )** cannot map overlapping ranges in a file. The virtual address range reserved for a file range is determined at the time of the initial mapping of the file range into a process address space. The system allocates only the virtual address range necessary to represent the initial mapping. As long as the initial mapping exists, subsequent attempts to map a different file range that includes any portion of theinitial range may fail with an **ENOMEM** error if an extended contiguous address range that preserves the mappings of the initial range cannot be allocated.
- Separate calls to **mmap( )** to map contiguous ranges of a file do not necessarily return contiguous virtual address ranges. The system may allocate virtual addresses for each call to **mmap( )** on a first available basis.
- The use of **MAP_FIXED** is strongly discouraged because it is not portable, and it may prevent the system from optimally allocating virtual address space.

The following combinations of protection modes are supported by PRO conformant systems:

PROT_NONE
PROT_READ
PROT_READ|PROT_EXECUTE
PROT_READ|PROT_WRITE
PROT_READ|PROT_WRITE|PROT_EXECUTE

NAME

ptrace( ) - process trace

SYNOPSIS

```
#include< sys/ptrace.h>

int ptrace(
        int request,
        pid_t pid,
        int addr,
        int data,
        int addr2
);
```

REMARKS

Much of the functionality of this capability is highly dependent on the underlying hardware. An application that uses this system call should not be expected to be portable across architectures or implementations.

DESCRIPTION

ptrace( ) provides a means by which a process can control the execution of another process. Its primary use is for the implementation of breakpoint debugging. The traced process behaves normally until it encounters a signal (see **signal**( ) for the list), at which time it enters a stopped state and the tracing process is notified via **waitpid**( ). When the traced process is in the stopped state, the tracing process can examine and modify the "core image" using **ptrace**( ). Also, the tracing process can cause the traced process either to terminate or continue, with the possibility of ignoring the signal that caused it to stop.

The request argument determines the precise action to be taken by **ptrace**( ) and is one of the following:

PT_SETTRC This request must be issued by a child process if it is to be traced by its parent. It turns on the child's trace flag which stipulates that the child should be left in a stopped state upon receipt of

a signal rather than the state specified by func; see **signal( )**. The *pid*, *addr*, *data*, and *addr2* arguments are ignored, and a return value is not defined for this request. Peculiar results occur if the parent does not expect to trace the child.

The remainder of the requests can only be used by the tracing process. For each, *pid* is the process ID of the process being traced, which must be in a stopped state before these requests are made.

PT_RIUSER, PT_RDUSER With these requests, the word at location *addr* in the address space of the traced process is returned to the tracing process. The **PT_RIUSER** command is used to read the traced program's text, while the **PT_RDUSER** command is used to read the traced program's data. The *addr* parameter must identify the start of a word in the traced process. The *data* and *addr2* arguments are ignored.

PT_WIUSER, PT_WDUSER With these requests, the value given by the *data* argument is written into the address space of the traced process at location *addr* which must be the address of the start of a word in the traced process's address space. Request **PT_WISUER** writes a word into text, while request **PT_WDUSER** writes a word into *data*. Upon successful completion, the value written into the address space of the traced process is returned to the tracing process. The *addr2* argument is ignored. These two requests may fail if *addr* is not the start address of a word, or if *addr* is a location in a pure procedure space and either another process is executing in that space or the tracing process does not

	have write access for the executable file corresponding to that space. Upon failure a value of -1 is returned to the tracing process and its **errno** is set to **EIO**.
PT_CONTIN	This request causes the traced process to resume execution. If the *data* argument is 0, all pending signals, including the one that caused the traced process to stop, are cancelled before it resumes execution. If the *data* argument is a valid signal number, the traced process resumes execution as if it had incurred that signal, and any other pending signals are canceled. The *addr* argument must be equal to 1 for this request. The *addr2* argument is ignored. Upon successful completion, the value of *data* is returned to the tracing process. This request fails if *data* is not 0 or a valid signal number, in which case a value of -1 is returned to the tracing process and its **errno** is set to EIO.
PT_EXIT	This request causes the traced process to terminate. The *addr*, *data*, and *addr2* arguments are ignored.
PT_SINGLE	This request causes a flag to be set so that an interrupt occurs upon the completion of one machine instruction, and then executes the same steps as listed above for request **PT_CONTIN**. If the processor does not provide a trace bit, this request returns an error. This effectively allows single stepping of the traced process. Whether or not the trace bit remains set after this interrupt is a function of the hardware.
PT_ATTACH	This request stops the process identified by *pid* and allows the calling process to trace it.

Process *pid* does not have to be a child of the calling process, but the effective user ID of the calling process must match the real and saved uid of process *pid* unless the effective user ID of the tracing process is super-user. The calling process can use the **waitpid( )** system call to wait for process *pid* to stop. The *addr, data,* and *addr2* arguments are ignored. It should be noted that the process to be traced does not have to execute a **PT_SETTRC** before being traced.

PT_DETACH This request detaches the traced process *pid* and allows it to continue its execution in the manner of **PT_CONTIN.**

To forestall possible fraud, **ptrace( )** inhibits the set-user-ID facility on subsequent **exec( )** calls. If a traced process calls **exec( )**, it stops before executing the first instruction of the new image showing signal **SIGTRAP.**

PT_RUREGS With this request, the word at location *addr* in the save_state structure at the base of the per-process kernel stack is returned to the tracing process. *addr* must be word-aligned and its size must not exceed the product of the system's stack size multiplied by the number of byters per page. The save_state structure contains the registers and other information about the process. (Please refer to the PRO ABI for information on registers and other process related information.) The *data* and *addr2* arguments are ignored.

PT_WUREGS The save_state structure at the base of the per-process kernel stack is written as it is read with request **PT_RUREGS.** Only a few locations can be written in this way: the general registers, most floating-point registers, a few control

registers, and certain bits of the interruption processor status word. The *addr2* argument is ignored.

PT_RDTEXT, PT_RDDATA These requests are identical to **PT_RIUSER** and **PT_RDUSER**, except that the *data* argument specifies the number of bytes to read and the *addr2* argument specifies where to store that data in the tracing process.

PT_WRTEXT, PT_WRDATA These requests are identical to **PT_WIUSER** and **PT_WDUSER** except that the *data* argument specifies the number of bytes to write and the *addr2* argument specifies where to read that data in that tracing process.

ERROR

In general, **ptrace( )** fails if any of the following conditions are encountered:

[EIO] *request* is an illegal number.

[EPERM] The specified process cannot be attached for tracing.

[ESRCH] *pid* identifies a process to be traced that does not exist or has not executed a **ptrace( )** with *request* **PT_SETTRC**.

If the *addr* argument to a **PT_CONTIN** or **PT_SINGLE** request is not 1, the Instruction Address Offset Queue (program counter) is loaded with the values *addr* and *addr+4* before execution resumes. Otherwise, execution resumes from the point where it was interrupted.

If the *addr* argument to a **PT_DETACH** request is not 1, the Instruction Address Offset Queue is loaded with the values *addr* and *addr2*.

NAME

ptsname( ) - get the name of a slave pty

SYNOPSIS

char *ptsname(int *fildes*);

REMARKS

ptsname( ) is useful only on systems that follow the **insf**( ) naming conventions for ptys.

DESCRIPTION

The passed parameter, *fildes*, is a file descriptor of an opened master pty. **ptsname**( ) generates the name of the slave pty corresponding to this master pty. This means that their minor numbers will be the same.

ERRORS

ptsname( ) fails and returns a NULL pointer under the following conditions:

- File descriptor does not refer to an open master pty.
- Request falls outside pty name-space.
- Pty device naming conventions have not been followed.
- **ptsname**( ) failed to find a match.

RETURN VALUE

Upon successful completion, **ptsname**( ) returns a string containing the full path name of a slave pty. Otherwise, a **NULL** pointer is returned. The return value may point to static data which is overwritten with each call to **ptsname**( ), so it should be copied if it is to be saved.

EXAMPLES

The following example gets the path of a slave pty corresponding to a master pty obtained through a pty clone open.

```
int fd_master;
char *path;
  ...
  fd_master = open("/dev/ptym/clone", O_RDONLY);
  path = ptsname(fd_master);
```

NAME

sbrk( ) - change data segment space allocation

SYNOPSIS

```
#include<inistd.h>
int brk(void *addr);
void *sbrk(int incr);
```

REMARKS

Clarification: PRO recommends the use of **malloc( )** to obtain additional working space. It is recognized, however, that many alternate malloc libraries rely on **sbrk** as an enabler.

DESCRIPTION

addr Points to the effective address of the maximum available data.

incr Specifies the number of bytes to be added to the current break. The value of *incr* may be positive or negative.

sbrk( ) is used to dynamically change the amount of space allocated for the calling process's data segment; see **exec( )**. The change is made by resetting the process's break value and allocating the appropriate amount of space.

When a program begins execution via **exec( )** the break is set at the highest location defined by the program and data storage areas. Typically, only programs with growing data areas need to use **sbrk( )**.

sbrk( ) adds *incr* bytes to the break value and changes the allocated space accordingly. *incr* can be negative, in which case the amount of allocated space is decreased. If **sbrk( )** is initially called with an *incr* of 0, then the value returned is the base of the existing data segment allocation.

When obtained, the data contents of the allocated region are undefined.

ERRORS

sbrk( ) will fail without making any change in the allocated space if the

following is true:

[ENOMEM] Such a change would result in more space being allocated than is allowed by a system-imposed maximum (see **ulimit**( )).

WARNINGS

The pointer returned by **sbrk**( ) is not necessarily word-aligned. Loading or storing words through this pointer could cause word alignment problems.

Be very careful when using **sbrk**( ) in conjunction with calls to the **malloc**( ) library routines. There is only one program data segment from which all three of these routines allocate and deallocate program data memory.

RETURN VALUE

Upon successful completion, **sbrk**( ) returns the old break value. Otherwise, **sbrk**( ) returns a value of -1, and **errno** is set to indicate the error.

NAME

shl_load( ), shl_definesym( ), shl_findsym( ), shl_gethandle( ), shl_getsymbols( ), shl_unload( ), shl_get( ) - explicit load of dynamic libraries

SYNOPSIS

```
#include <dl.h>
shl_t shl_load(const char *path, int flags, long address);
int shl_findsym(
    shl_t *handle,
    const char *sym,
    short type,
    void *value
);

int shl_definesym(
    const char *sym,
    short type,
    long value,
    int flags
);

int shl_getsymbols(
    shl_t handle,
    short type,
    int flags,
    void *(*memory) ( ),
     struct shl_symbol **symbols,
);

int shl_unload(shl_t handle);
int shl_get(int index, struct shl_descriptor **desc);
int shl_gethandle(shl_t handle, struct shl_descriptor **desc);
```

DESCRIPTION

These routines can be used to programmatically load and unload dynamic libraries, and to obtain information about the libraries (such as the addresses of symbols defined within them). The routines themselves are accessed by specifying the **-ldld** option on the command line with the **c89** command.

Dynamic libraries must be created using position independent code. (See the PRO ABI for further information.)

shl_load( ) Attaches the dynamic library named by *path* to the process. The library is mapped at the specified address. If *address* is 0L, the system chooses an appropriate address for the library. This is the recommended practice because the system has the most complete knowledge of the address space (see **DEPENDENCIES**). The *flags* argument is made up of several fields. One of the following must be specified:

BIND_IMMEDIATE Resolve symbol references when the library is loaded.

BIND_DEFERRED Delay code symbol resolution until actual reference.

Zero or more of the following can be specified by doing a bitwise OR operation:

BIND_FIRST Place the library at the head of the symbol search order.

BIND_NONFATAL Default **BIND_IMMEDIATE** behavior is to treat all unsatisfied symbols as fatal. This flag allows binding of unsatisfied code symbols to be deferred until use.

BIND_NOSTART Do not call the initializer for the dynamic library when the library is loaded, nor on a future call to **shl_unload( )**.

BIND_VERBOSE Print verbose messages concerning possible unsatisfied symbols.

BIND_RESTRICTED Restrict symbols visible by the library to those present at library load time.

DYNAMIC_PATH Allow the loader to dynamically search for the library specified by the *path* argument. (See the PRO ABI for further information.)

If successful, **shl_load( )** returns a handle which can be used in subsequent calls to **shl_findsym( )**, **shl_unload( )**, or **shl_gethandle( )**; otherwise **NULL** is returned.

shl_findsym( ) Obtains the address of an exported symbol *sym* from a dynamic library. The *handle* argument should be a pointer to the handle of a loaded dynamic library that was returned from a previous call to **shl_load( )** or **shl_get( )**. If a pointer to **NULL** is passed for this argument, **shl_findsym( )** searches all currently loaded dynamic libraries to find the symbol; otherwise **shl_findsym( )** searches only the specified dynamic library. The return value of *handle* will be **NULL** if the symbol found was generated via **shl_definesym( )**. Otherwise the handle of the library where the symbol was found is returned. The special handle **PROG_HANDLE** can be used to refer to the program itself, so that symbols exported from the program can also be accessed dynamically. The *type* argument specifies the expected type for the symbol, and should be one of the defined constants **TYPE_PROCEDURE, TYPE_DATA**, or **TYPE_UNDEFINED**. The latter value suppresses type checking. The address of the symbol is returned in the variable pointed to by *value*. If a dynamic library contains multiple versions of the requested symbol, the latest version is returned. This routine returns 0 if successful; otherwise -1 is returned.

shl_definesym( ) Adds a symbol to the dynamic library symbol table for the current process making it the most visible definition. If the value falls in the range of a currently loaded library, an association will be made and the symbol is undefined once the associated library is unloaded. The defined symbol can be overridden by a subsequent call to this routine or by loading a more visible library that provides a definition. Symbols overridden in this manner may become visible again if the overriding definition is removed.

Possible symbol types include:

TYPE_PROCEDURE	Symbol is a function.
TYPE_DATA	Symbol is data.

At the present time, no flag values have been defined. It is recommended that the flag value be set to zero to prevent conflicts with future uses of this flag.

The use of **shl_definesym( )** to redefine an initializer is not supported

shl_getsymbols( ) Provides an array of symbol records, allocated using the supplied memory allocator, that are associated with the library specified by *handle*. If the *handle* argument is a pointer to **NULL**, symbols defined using **shl_definesym( )** are returned. If multiple versions of the same symbol have been defined within a library or with **shl_definesym( )**, only the version from the specified symbol information source that would be considered for symbol binding is returned. The type argument is used to restrict the return information to a specific type. Values of **TYPE_PROCEDURE** and **TYPE_DATA** can be used to limit the returned symbols to be either code or data respectively. The constant **TYPE_UNDEFINED** can be used to return all symbols, regardless of type. The *flags* argument must have one of the following values:

IMPORT_SYMBOLS	Return symbols found on the import list.

EXPORT_SYMBOLS	Return symbols found on the export list. All symbols defined via **shl_definesym**( ) are export symbols.
INITIALIZERS	Return symbols from the dynamic library initializer list.

Zero or more of the following can be specified by doing a bitwise OR operation:

NO_VALUES	Only makes sense when combined with **EXPORT_SYMBOLS** or **INITIALIZERS**. Do not calculate the value field in the shl_symbol structure. The value field will contain an undefined value. Not to be used with **GLOBAL_VALUES**.
GLOBAL_VALUES	Used with **EXPORT_SYMBOLS** and **INITIALIZERS**, this flag causes shl_getsymbols( ) to return the most visible occurrence, and to set the *value* and *handle* fields of the shl_symbol structure. Not to be used with **NO_VALUES**.

The memory argument should point to a function with the same interface as **malloc**( ).

The return information consists of an array of the following records (defined in **<dl.h>**):

```
struct shl_symbol {
        char *name;
        short type;
        void  *value;
        shl_t handle;
};
```

The type field in the return structure can have the values **TYPE_PROCEDURE**, **TYPE_DATA**, or **TYPE_STORAGE**, where **TYPE_STORAGE** is a subset of **TYPE_DATA**. The value and handle fields are only valid if initializers or export symbols are requested and the **NO_VALUES** flag is not specified. The *value* field contains the address of the symbol, while the *handle* field is the handle of the library that defined the symbol, or **NULL** for symbols defined via the **shl_definesym( )** routine and is useful in conjunction with the **GLOBAL_VALUES** flag.

If successful, **shl_getsymbols( )** returns the number of symbols found; otherwise it returns -1.

shl_unload( ) Can be used to detach a dynamic library from the process. The handle argument should be the handle returned from a previous call to **shl_load( )**. **shl_unload( )** returns 0 if successful; otherwise -1 is returned. All explicitly loaded libraries are detached automatically on process termination.

shl_get( ) Returns information about currently loaded libraries, including those loaded implicitly at startup time. The index argument is the ordinal position of the dynamic library in the dynamic library search list for the process. A subsequent call to **shl_unload( )** decrements the index values of all libraries having an index greater than the unloaded library. The index value -1 refers to the dynamic loader. The *desc* argument is used to return a pointer to a statically allocated buffer containing a descriptor for the dynamic library. The buffer for the descriptor used by **shl_get( )** is static; the contents should be copied elsewhere before a subsequent call to the routine. The routine returns 0 normally, or -1 if an invalid index is given.

shl_gethandle( ) Returns information about the library specified by the *handle* argument. The special handle **PROG_HANDLE** can be used to refer to the program itself. The descriptor returned is the

same as the one returned by the **shl_get**( ) routine. The buffer for the descriptor used by **shl_gethandle**( ) is static; the contents should be copied elsewhere before a subsequent call to the routine. The routine returns 0 normally, or -1 on error.

DIAGNOSTICS

If a library cannot be loaded, **shl_load**( ) returns **NULL** and sets **errno** to indicate the error. All other functions return -1 on error and set **errno**. If **shl_findsym**( ) cannot find the indicated symbol, **errno** is set to zero. If **shl_findsym**( ) finds the indicated symbol but cannot resolve all the symbols it depends on, **errno** is set to **ENOSYM**.

If a call to **shl_load**( ) or **shl_findsym**( ) fails with **ENOSYM**, the process may be left in an inconsistent state. Some symbol resolutions may have occurred before the failure, and these may be invalid. The program should probably be terminated if this occurs.

ERRORS

Possible values for **errno** include:

[ENOEXEC]	The specified file is not a dynamic library, or a format error was detected.
[ENOSYM]	Some symbol required by the dynamic library could not be found.
[EINVAL]	The specified handle or index is not valid or an attempt was made to load a library at an invalid address.
[ENOMEM]	There is insufficient room in the address space to load the library.
[ENOENT]	The specified library does not exist.
[EACCES]	Read or execute permission is denied for the specified library.

WARNINGS

shl_unload( ) detaches the library from the process and frees the memory

allocated for it, but does not break existing symbolic linkages into the library. In this respect, an unloaded dynamic library is much like a block of memory deallocated via **free**( ).

Some implementations may not, by default, export all symbols defined by a program (instead exporting only those symbols that are imported by a dynamic library seen at link time).

All symbol information returned by **shl_getsymbols**( ), including the *name* field, become invalid once the associated library is unloaded by **shl_unload**( ).

DEPENDENCIES

The only value for the address field is 0L. Any other value is treated as if it had been specified as 0L.

NAME

shmat( ) - shared memory operations

SYNOPSIS

#include <sys/shm.h>

char *shmat(int *shmid*, void **shmaddr*, int *shmflg*);

REMARKS

Clarification: If the shared memory segment is not already attached, *shmaddr* must be specified as zero and the segment is attached at a location selected by the operating system. That location is identical in all processes accessing that shared memory object.

If the shared memory segment is already attached, a non-zero value of *shmaddr* is accepted, provided the specified address is identical to the current attach address of the segment.

Note that alternative interfaces for interprocess communication are being developed by industry standards bodies. Application developers are encouraged to to implement their software so that it may be easily modified to apply future standard methods for IPC.

ERRORS

shmat( ) fails and returns -1 if any of the following conditions are encountered:

[EINVAL]	*shmaddr* is not zero and the machine does not permit non-zero values or *shmaddr* is not equal to the current attach location for the shared memory segment.
[EINVAL]	*shmaddr* is not the data segment start address of a shared memory segment.
[EINVAL]	The calling process is already attached to *shmid*.

READER COMMENTS

Application Programming Interface for PA-RISC Systems

Edition X August 1993

Please use this Reader Comment Card to evaluate this document and tell us of problems or suggest improvements.

Please rate the quality of each item below in terms of your expectations:

	Far Below Expectations	Below Expectations	Meets Expectations	Exceeds Expectations	Far Exceeds Expectations
Retrievability	1	2	3	4	5
Table of Contents	1	2	3	4	5
Headings in Chapters	1	2	3	4	5
Appendices	1	2	3	4	5
Organization	1	2	3	4	5
Completeness	1	2	3	4	5
Accuracy	1	2	3	4	5
Readability	1	2	3	4	5
Language Usage	1	2	3	4	5
Layout	1	2	3	4	5

Recommended improvements (attach additional information if needed:)

_____

_____

_____

_____

Name: Company:
Job Title: Address:
Phone:
Please enter your system name and series number, e.g. HP 9000 series 700:_____

PRO has the right to use submitted suggestions without obligation, with all such ideas becoming property of the Precision Risc Organization.

Return to: **Precision Risc Organization**
19111 Pruneridge Avenue M/S 44MU
Cupertino, CA 95014